Java Programming

UNIT I

Fundamentals of Object-Oriented Programming: Object-Oriented Paradigm – Basic Concepts of Object-Oriented Programming – Benefits of Object-Oriented Programming – Application of Object-Oriented Programming. Java Evolution: History – Features – How Java differs from C and C++ – Java and Internet – Java and www –Web Browsers. Overview of Java: simple Java program – Structure – Java Tokens – Statements – Java Virtual Machine.

UNIT II

Variables & Control Structures

Constants, Variables, Data Types - Operators and Expressions – Decision Making and Branching: if, if...else, nested if, switch? : Operator - Decision Making and Looping: while, do, for – Jumps in Loops - Labeled Loops – Classes, Objects and Methods.

UNIT III

Arrays & Classes

Arrays, Strings and Vectors – Interfaces: Multiple Inheritance – Packages: Putting Classes together – Multithreaded Programming.

UNIT IV

Error Handling & Graphics

Managing Errors and Exceptions – Applet Programming – Graphics Programming.

UNIT V

I/O Streams

Managing Input / Output Files in Java: Concepts of Streams- Stream Classes – Byte Stream classes – Character stream classes – Using streams – I/O Classes – File Class – I/O exceptions – Creation of files – Reading / Writing characters, Byte-Handling Primitive Data Types – Random Access Files.

UNIT-I

Fundamentals of Object Oriented programming: Object oriented paradigm – Basic concepts of Object oriented programming-Benefits of Object oriented programming – Application of Object oriented programming. **Java Evolution :** History – Features – How Java differs from C and C++ - Java and Internet- Java and www – Web Browsers. **Overview of Java:** Simple java program structure- java tokens – statements – java virtual machine.

1.1 Fundamentals of Object oriented programming:

1.1.1 Introduction

Since the invention of the computers, many programming approaches have been tried. These include techniques such as modular programming, top-down programming, bottom-up programming, and structured programming. The primary motivation in each case has been the concern to handle the increasing complexity of programs that are reliable and maintainable. These techniques became popular among programmers over the last two decades.

Object-oriented programming (OOP) is an approach to program organization and development, which attempts to eliminate some of the pitfalls of conventional programming methods by incorporating the best of structured programming features with several new concepts.

1.1.2 Object Oriented paradigm

- Object oriented paradigm emphasis on data rather than procedure
- Programs are divided into what are know as objects
- Data structure are designed such that they characterize the objects
- Methods that operate on the data of an object are typed together in the data structure
- Hidden data cannot be accessed by external function
- Objects may communicate with each other through methods
- New data and methods can be easily added whenever necessary
- Follow bottom-up approach in program design

Object-oriented programming is an approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand.

1.3 Basic concepts of object-oriented programming

Objects and classes :

Objects are the basic runtime entities in an object-oriented system.

A **class** is thus a collection of objects of similar type. For example, mango, apple and orange are members of the class fruit.

Data Abstraction and Encapsulation :

The wrapping up of data methods into single unit (called class) is known as **Encapsulation**. Data encapsulation is the most striking feature of a class. The data is not accessible to the outside world and only those methods, which are wrapped in the class, can access it. This insulation of the data from direct access by the program is called data hiding.

Abstraction refers to the act of representing essential features without including the background details or explanation. Classes use the concepts of abstraction and are defined as a list of abstract attribute such as size, weight and cost, and methods that operate on these attributes.

Polymorphism

Polymorphism means the ability to take more than one form

Polymorphism is a feature that allows one interface to be used for a general class of actions.

For example a single button of your mobile phone can be used to dial number or send or message or even take picture.

This helps to reduce complexity by allowing same interface to be used to specify a general class of actions.

Inheritance

Inheritance is the process by which the objects of one class acquire the property of object of another class. In OOP, concept of inheritance provides the idea of reusability. This means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing one.

Dynamic Binding

Binding refers to the linking of a procedures call to the code to be executed in response to the call. Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at run time. It is associated with polymorphism and inheritance.

Message communication

An object-oriented program consists of a set of objects that communicates with each other.

1. Creating classes that define object and behaviors.
2. Creating objects from class definitions.
3. Establishing communication among objects.

Objects communicates with one another by sending and receiving information much the same way as people pass messages to one another.

Eg: Employee. Salary (name) ;

1.1.4 Benefits of OOP

- Through inheritance, we can eliminate redundant code and extend the use of existing classes
- We can build programs from the standard working modules that communicates with one another, rather than having to start writing the code from scratch. This leads to saving of development time and higher productivity.
- The principle of data hiding helps the programmer to built secure programs that cannot be invaded by code in other parts of the program.
- It is possible to have multiple objects to coexist without any interference.
- It is possible to map objects in the problem domain to those objects in the program.
- It is easy to portion the network the work in a project based on objects.
- The data-centered design approach enables us to capture more details of a model in a implementable form.
- Object-oriented systems can be easily upgraded from small to large systems.
- Message passing techniques for communication between objects make the interface descriptions with external systems much simpler.
- Software complexity can be easily managed.

1.1.5 Application of OOP

Real-business systems are often much more complex and contain many more objects with complicated attributes and methods.

- Real-time systems
- Simulations and modeling
- Object-oriented databases
- Hypertext, hypermedia and expertext
- AI and experts systems
- Neural networks and parallel programming
- Decisions support and office automation systems.
- CIM/CAD/CAD system

It is believed that the richness of OOP environment will enable the software industry to improve not only the quality of software systems but also its productivity.

2. JAVA EVOLUTION:

1.2.1 JAVA HISTORY

Java is a general purpose object oriented programming language developed by Sun Microsystems of USA in 1991.

Java Milestones

Year	Development
1990	Sun Microsystems decided to develop special software that could be used to manipulate consumer electronic devices .A team of Sun Microsystems programmers headed by James Gosling was formed to undertake this task.
1991	After exploring the possibility of using the most popular object-oriented language C++, the team announced a new language named "OAK".
1992	The team known as Green Project team by Sun, demonstrated the application of their new language to control a list of home appliances using a hand-held device with a tiny touch-sensitive screen.
1993	The World Wide Web appeared on the Internet and transformed the text-based internet

	into a graphical-rich environment. The Green project team came up with the idea of developing Web applets(tiny programs) using the new language that could run on all types of computers connected to Internet.
1994	The team developed a Web browser called "HotJava" to locate and run applet programs on internet. HotJava demonstrated the power of the new language, thus making it instantly popular among the Internet users.
1995	Oak was renamed "Java", due to some legal snags. Java is just a name and is not an acronym. Many popular companies including Netscape and Microsoft announced their support to Java.
1996	Java established itself not only as a leader for Internet programming but also as a general-purpose, object-oriented programming language, Sun releases Java Development Kit 1.0.
1997	Sun releases Java Development Kit 1.1(JDK1.1).
1998	Sun releases the Java 2 with version 1.2 of the software Development Kit(SDK1.2)
1999	Sun releases Java 2 platform . Standard Edition(J2SE) and Enterprise Edition(J2EE).
2000	J2SE with SDK1.3 was released.
2002	J2SE with SDK1.4 was released.
2004	J2SE with JDK 5.0 (instead of JDK 1.5) was released. This is known as J2SE 5.0.

The most striking feature of the language is that it is a platform-neutral language. Programs developed in Java can be executed anywhere on any system.

1.2.2 JAVA FEATURES

The inventors of Java wanted to design a language which could offer solutions to some of the problems encountered in modern programming. Java features are.

- Compiled and Interpreted.
- Platform-Independent and Portable.
- Object-Oriented.
- Robust and secure
- Distributed
- Familiar, Simple and Small.
- Multithreaded and Interactive.

- High Performance.
- Dynamic and Extensible.
- Ease of Development.
- Scalability and Performance.
- Monitoring and Manageability.
- Desktop Client

Compiled and Interpreted

Usually a computer language is either compiled or interpreted. Java combines both these approaches. First, Java compiler translates source code into *bytecode* instructions. Bytecodes are not machine instructions and therefore, in the second stage, Java interpreter generates machine code that can be directly executed by the machine that is running the Java program. So Java is both a compiled and an interpreted language.

Platform-Independent and Portable

The most significant contribution of Java over other languages is its portability. Java programs can be easily move from one computer to another. Changes and upgrades in operating systems and system resources will not force any changes in Java programs. This is the reason for popularity of language as programming on Internet. We ca download a Java applet from a remote computer onto our local system via Internet and execute it locally.

Java ensures portability in two ways. First, Java compiler generates bytecode instructions that can be implemented on any machine. Secondly, the size of the primitive data types are machine-independent.

Object-Oriented

Java is a true object-oriented language. Almost everything in Java is an object. All program code and data reside within objects and classes. Java comes with an extensive set of classes, arranged in packages, that we can use in our programs by inheritance. The object model in Java is simple and easy to extend.

Robust and Secure

Java is a robust language. It provides many safeguards to ensure reliable code. It has strict compile time and run time checking for data types. Java also incorporates the concepts of exception handling which captures series errors and eliminates any risk of crashing the system.

Security becomes an important issue for a language that is used for programming on Internet. Threat of viruses and abuse of resources are everywhere. Java systems not an applet. The absence of pointers in Java ensures that programs cannot gain access to memory locations without proper authorization.

Distributed

Java is designed as a distributed language for creating applications on networks. It has the ability to share both data and programs. Java applications can open and enables multiple programmers at multiple remote locations to collaborate and work together on a single project.

Simple, Small and Familiar

Java is a small and simple language. Many features of C and C++ that are either redundant or sources of unreliable code are not part of Java.

For example, Java does not use pointers, preprocessor header files, goto statement and many others. It also eliminates overloading and multiple inheritance.

Familiarity is another striking feature of Java. To make the language look familiar to the existing programmers, it was modeled on C and C++ languages. Java is a simplified version of C++.

Multithreaded and Interactive

Multithreaded means handling multiple tasks simultaneously. Java supports multithreaded programs. This means that we need not wait for the application to finish one task before beginning another.

The Java runtime comes with tools that support multiprocess synchronization and construct smoothly running interactive systems.

High Performance

Java performance is impressive for an interpreted language, mainly due to the use of intermediate bytecode. Java speed is comparable to the native C/C++. Java architecture is also designed to reduce overheads during runtime. Further, the incorporation of multithreading enhances the overall execution speed of Java programs.

Dynamic and Extensible

Java is a dynamic language. Java is capable of dynamically linking in new class libraries, methods and objects. Java can also determine the type of class through a query, making it possible to either dynamically link or abort the program, depending on the response.

Java programs support functions written in other languages such as C and C++. These functions are known as native methods.

Ease of Development

Java 2 Standard Edition (J2SE) 5.0 supports features, such as Generics, Enhanced for Loop, Autoboxing or unboxing, Typesafe Enums, Varags, Static import and Annotation. These features reduce the work of the programmer by shifting the responsibility of creating the reusable code to the compiler. The resulting source code is free from bugs. Each of the linguistic features is designed to develop Java programs in an easier way.

Scalability and Performance

J2SE 5.0 assures a significant increase in scalability and performance by improving the startup time and reducing the amount of memory used in Java 2 runtime environment

Monitoring and Manageability

Java supports a number of APIs, such as JVM Monitoring and Management API, Sun Management Platform Extension, Logging, Monitoring and Management Interface, and Java Management Extension(JMX) to monitor and manage Java applications .

Desktop client

J2SE 5.0 provides enhanced features to meet the requirements and challenges of the Java desktop users. It provides an improved Swing look and feel called Ocean. This feature is mainly used for developing graphics applications that require OpenGL hardware acceleration.

1.2.3 How Java differs from C and C++ :

C	JAVA
Size of type definition (statement C keyword)	Not supported
Struct, union(datatype)	Not supported
Auto, extern, register, signed unsigned (type modifier key word)	Not supported
Pointers	Not supported
Not supported	New operator like instance of and >>>
Not supported	Labeled, break, continue

C++	JAVA
Operator overloading	Not supported
Template class	Not supported
Multiple inheritance	Interface
Pointer	Not supported
Destructor	Destructor replaced as a Finalize() function

1.2.4 Java and Internet

Java is strongly with the Internet because of the fact that the application program written in Java was HotJava, a web browser to run Java applets on Internet. Internet users can use Java to create applet programs and run them locally using a "Java-enabled browser" such as HotJava . They can also use a Java-enabled browser to download an applet located on a computer anywhere in the internet and run it on his local computer. In fact java applets have made the Internet a true extension of the storage system of the local computer.

Internet users can also set up their Websites containing Java applets that could be used by other remote users of Internet. The ability of Java applet to hitch a ride on the Information Superhighway as made Java a unique programming language for the Internet. In fact, due to this, Java is popularly known as Internet language

1.2.5 Java and World Wide Web

World Wide Web(WWW) is an open-ended information retrieval system designed to be used in the Internet's distributed environment. This system contains what are known as Web pages that provide both information and controls. Unlike a menu-driven system where we are guided through a particular direction using a decision tree structure, the Web system is open-ended and we can navigate to a new document in any direction .

Before Java, the World Wide Web was limited to the display of still images and texts. However, the incorporation of Java into Web pages has made it capable of supporting animation, graphics, games and wide range of special effects. With support of Java, the web has become more interactive and dynamic.

Java communicates with a Web page through a special tag called <APPLET>. The following are the communication steps.

- The user sends a request for an HTML document to the remote computer's Web server is a program that accepts a request, processes the request, and sends the required document.

- The HTML document is returned to the user's browser. The document contains the APPLET tag, which identifies the applet.
- The corresponding applet byte code is transffered to the user's computer. This byte code had been previously created by the Java source code file for that applet.
- The Java-enabled browser on the user's computer interprets the byte codes and provides output.
- The user may have further interaction with the applet but with no further downloading from the provider's Web server. This is because the byte code contains all the information necessary to interpret the applet.

1.2.6 Web Browsers

A large portion of the Internet is organized as the World Wide Web which uses hypertext. Web browsers are used to navigate through the information found on the net. They allow us to retrieve the information spread across the Internet and display it using the hypertext markup language (HTML).

Among other Web browsers, includes

- HotJava
- Netscape Navigator
- Internet Explorer

Hot Java

HotJava is the Web browser from Sun Microsystems that enables the display of interactive content on the Web, using the Java language. HotJava is written entirely in Java and demonstrates the capabilities of Java programming language.

When the Java language was first developed and ported to the Internet, no browsers were available that could run Java applets. HotJava is currently available for the SPARC/Solaris platform as well as Windows 95, Windows NT and Windows XP.

Netscape Navigator

Netscape Navigator was from Netscape Communications Corporation, is a general purpose browser that can run Java applets. It has many useful features such as visual display about downloading process and indication of number of bytes downloaded. It also supports JavaScript, a scripting language used in HTML documents.

Internet Explorer

Internet Explorer is another popular browser developed by Microsoft for Windows 95, NT and XP workstations. Both the Navigator and Explorer use tool bars, icons, menus and dialog boxes for easy navigation. Explorer uses a just-in-time (JIT) compiler which increases the speed of execution.

1.2.7 Hardware and Software Requirements:

Currently supported on Windows95, WindowsNT, Windows XP,Sun Solaris, Macintosh & Unix machines. The minimum requirements

* IBM-Compatible 486 system
* Hard drive
* Minimum of 8 MB memory
* Windows 95
* CD-ROM
* Microsoft-compatible mouse
* Windows-Compatible sound card

1.3 OVER VIEW OF JAVA LANGUAGE

1.3.1 INTRODUCTION :

We have developed two types of java program:

1. Stand-alone applications.
2. Web applications.

The executing Java stand alone applications involve two steps:

1. Compiling source code in to byte code using javac compiler.
2. Executing the byte code program using java interpreter.

1.3.2 Simple Java Program:

- Java program source code is created with a simple text editor and saved as text file with the extension java.
- The code as well as the name of the program is case sensitive.
- For example

To print my first java program. Save as program

```
Public class program
{
Public static void main (string a [])
{
System.out. println ("My first java program");
}
}
```

Public :

Public is an access modifier used to define access restrictions. In this case, public implies that the class program is publicity availability and its function main ( ) can be accessed by any one.

Class:-

This keyword is used to define a class and precedes in the user defined class name i.e. program in this program.

Static:-

This keyword state that the function main ( ) is a class function which can be caved without instantiating class.

All static functions have only one copy to be used for all instances of the cass.

Void:-

The void refers to the return type and shows that the function mann ( ) does not return any value.

String a[]:-

This is the parameter list of function main ( )

It says that the main ( ) function can accept an away of string objects that represents the come – line arguments passed by the user at the time of execution.

System.out.println:-

- System is a predefined class to access the keyboard and monitor. It defines various data members and methods.
- Out property refers to the console/ monitor. It defines an output stream used to output something on the monitor.
- Print in ( ) is the function of this output stream and is used to print different types of v ariables and text.

{ }:

These braces are used to define class body and method body.

Comments:

// -single line comment

/* */-Multiline comment

Compilation and Execution:-

- After the creation of source program, the next step to compile this file.
- The JDK includes Java, exe, which is used to compile java program.
- Then set the path of this executable file in the PATH environment variable.
- If we have installed JDK at c: and it root directory is J2sdk 1.4.0 bin ; % path %
- After providing the path of compiler and other tools, change to the folder in which you saved your java source file in the command prompt window.
- Then type the following statement at the command prompt:

Java C program .java

If you do not get any messages, it means the complier successfully created a file name program class.

- Then run the java program. At the command prompt, type the following command to launch JVM Java program
- Then you will get the output

My first java program

1.3.3 Java Program Structure:

Document section
Package Statement
Import Statement
Interface Statements
Class Definitions

Main Method class
{
Main method definition
}

Document section:

- Comprises a set of comment line giving the name of the program, the author and other details,
- Comments must be explain Why & What of classes and how of algorithm.
- This would help in a maintaining the program
- /*…..*/ Document comment
- // single line comment.

Package statement:

Its include a package statement, like package packagename;

Import statements:

Include some import import statements eg: - import jva.io.*;

Interface Statements:

Include an interface concept. It's like a method.

Class definition:

- Classes are the primary and essential elements of java program.
- These classes are used to map the object of real-world problems.
- The number of classes used depends on the complexity of the problem.

Main Method Class:

- Main method as its starting point of the stand alone application.
- It's a essential part of java program.
- Java program contains only one main method.
- The main method creates objects of various classes and establishes communications b/w them.

1.3.4 Java Tokens:

Basically java is a collection of classes.

A class is defining by set of declaration statements and methods containing executable statements.

Most statements contain expressions which describe the actions carried out n data.

Smallest individual units in a program are known as tokens.

Java contains 5 types of tokens:

1. Reserved Keywords 2.Identifiers 3.Literals 4.Operators 5.Seperators

Java Character Set

✓ The smallest unit of java language are the characters used to write java tokens

✓ These characters are defined by the Unicode character set, an emerging standard that tries to create characters for a large number of scripts world wide.

✓ The Unicode is a 16 bit character coding system and currently support more than 34000 defined characters derived from 24languages from America, Europe, Middle East, Africa and Asia including India.

✓ However, most of us use only the basic ASCII characters, which include letters, digits and punctuations marks, used in normal English.

Key Words

✓ Key words are an essential part of language definition.
✓ They implement specific features of the language
✓ Java language has reserved 50 words as keywords
✓ These keywords combined with operators and separators according to syntax, from `definition of the java language.
✓ Understanding the meaning of all these words is important for java programmers.
✓ Java keywords have specific meaning in java.
✓ W cannot use them as names for variables, classes, methods and so and so.

Abstract, byte, class, do, extends, for, import, long, private, short, switch, throws, volatile, assert, case, const, double, final, goto, instanceof, native, protected, static, synchronized, transient, while, Boolean, catch, continue, else, finally, if, int, new, public, striftfp, this, try, break, char, default, enum, float, implements, interface, package, return, super, throw, void.

Identifiers

✓ These are programmer-designed tokens
✓ They are used for naming classes, methods, variables, objects, labels, packages and interfaces in a program.

Rules

- They can have alphabet, digit and the underscore and dollar sign characters
- They must not begin with a digit
- Upper case and lower case letter are distinct
- They can be of any length.

The identifier must be meaning full, short enough to quickly and easily typed and long enough to be descriptive and easily read.

- Names of all public methods and instance variables start with a leading lower case letter. EX: average, sum
- When more than one words are used in a name the second and subsequent words are marked with leading upper case letter. EX: day Temperature
- All private and local variables use only lower case letters combined with underscore. EX: batch strength

- All classes and interfaces start with a leading upper case letter(and each subsequent word with a leading upper case letter) EX: Student, HelloJava
- Variable that represent constant value used are upper case letters and underscore between words. EG: TOTAL, F_MAX.

Literals :

Literals in java are a sequence of characters (digits, letters and others) that represent constant values to be stored in a variable

Type of literals :

- Integer literals
- Floating-point literals
- Character literals
- String literals
- Boolean literals

Each of them has type associated with it. The types describes how the values be have and how they are stored

Operators

An operator is that takes one or more arguments and operates on them to produce a result.

Separators

Separators are simples used to indicate where group of code are divided and arranged

They basically define the shape and function of our code

Name	What it is used for
Parenthesis ( )	Used to enclose parameters in methods definition and I vocation, also used for defining precedence in expressions containing expression for flow control and surrounding cast types
Braces { }	Used to contain the values of automatically initialized arrays and

	to define a block of code for methods and local scopes
Brackets [ ]	Used to declare array types and for dereferencing array values
Semicolon ;	Used to separate statements
Comma ,	Used to separate consecutive identifiers in a variable declaration also used to change statements together inside a 'for' statement
Period .	Used to separate package names from sub-packages and classes also used to separate variable or method from a reference variable

1.3.5 JAVA STATEMENTS :

Java statements in java are like sentences in natural languages. Statement is an executable combination of tokens ending with a semicolon (;) mark.

Statement	Description	Remarks
Empty statements	These do nothing and are used during program development as place holder	Same as c and c++
Labeled statement	Any statement begin with label such labels must not be keywords already declared local variables or previously used labels in this modules. labels in java are used as the argument of jump statements which are described late I this list	Identical to c and c++ except their use with jump statement
Expression statement	Most statements are expression statements. Java has 7 types of expression statements. Assignment, pre-increment, pre-decrement, method call, and allocation expression.	Same as c++
Selection statement	These select one several control follows there are three types of selection statement in java, if , else if, switch	Same as c and c++
Iteration statements	This is specify how and when looping will take place. There are three type s of iteration statements while, do , for	Same as c and c++ except for jump and labels

Jump statements	Jump statements pass controls to the banging or end of the current block or to a labeled statement. Such labeled must be in the same block and continue labels must be on iteration statement the four type of jump statement are break , continue, return, and throw	C and c++ do not use labels with jump statements
Synchronization statements	These are used for handling of issues with multithreading	Now available in c and c++
Guarding statements	These are used for safe handling of code that may cause exception (such as division by 0). These statement s use the keywords try, catch and finally	Same as in c++ except finally statements.

1.3.6 Implementing a Java program:

It involves some steps:

1. Creating the program
2. Compiling the program
3. Running the program

In before we must installed JDK (Java Development Kit) our systems.

Creating the program:

Eg:

```
Class Test
{
   Public static void main(String args[])
    {
      System.out.println(“Hellow”);
      System.out.println(“Welcome to the world”)
           }
}
```

Here we must the save the program in file called Test.java, Class name and java file is same.

1.3.7 Command Line Arguments

- There may be occasions when we may like our program to act in a particularly way depending on the way input provided at the time of execution.
- This is achieved in java programs by using what are known as commend line arguments.
- Here java programs that can receives and use the arguments provided in the comment line.

Public static void main(String args[])

Here, args[] is declared as an array of string

Example :

```
Import java.io.*
Class ComLineTest  {
Public static void main(String args[])
        int count, i=0;
String string;
        count= args.length;
        System.out.println("Number of arguments ="+count);
        while(i< count)
          {
            string=args[i];
            i=i+1;
           System.out.println(i+":"+"java  is"+string+"!")
         }
}}
```

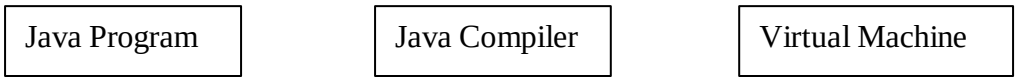
1.3.8 JAVA VIRTUAL MACHINE:

* The java compiler produces an intermedia code known as bytecode for a machine that does not exist. This Machine is called the Java Virtual Machine.

* It exists only onside the computer memory.

* It is simulated computer within the computer and does all major functions of a real computer.

* Java program into byte code which is also referred as Virtual Machine code.



Source code Byte code

Process of compilation

*The Virtual machine code is not machine specific. The machine specific code (known as machine code) is generated by the Java Interpreter by acting as an intermediary b/w the virtual machine and the real machine.

Virtual machine Real Machine

The java frame work act as the intermediately b/w the user and the JVM.

Suggested questions

1. **What are the java tokens?**
2. **Explain the basic concepts of OOPs**
3. **What are the applications and features of OOPs**
4. **Explain JVM.**
5. **What are the Java Statements.**

UNIT –II

Constants, Variables, Data types – Operators and Expressions-Decision Making and Branching : if , if –else, nested if ,switch , ?: operator – Decision Making and Looping : while , do, for – Jumps in Loops – Labeled Loops – Classes , Objects and Method.

2.1 INTRODUCTION

2.1.1 Constants:

Java has several constants:

Integer: its have 3 types of integer constants.

1. Decimal integers-set of digits (0-9, minus sign -) EG: 123, -123
2. Octal integer- combination of digits from the set 0-7 with a leading 0. EG: 0,037
3. Hexadecimal – digits proceeds by 0x, 0X, alphabets A-F. EG: 0x, 0x9F

Real Constants: -its represent quantities that continuously, such as a distances, heights, temperatures.

Single character constants: A single character constant contains a single character enclosed within a pair of single quote. Eg:'5','r'

String constants: A string is a sequence of characters enclosed b/w double quotes the characters may be alphabets, digits, special characters and blank space.Eg:"1997","Hi Ji"

Backslash Character String:

Java supports some special backslash character constants that are used in output methods.

A list of such backslash character combination of is known as escape sequence.

Constant	Meaning
'\b'	Back space
'\f'	From feed
'\n'	New line
'\r'	Carriage return
'\t'	Horizontal tab
'\''	Single quote
'\"'	Double quote
'\\'	Backslash

2.1.2 Variables:

Is an identifier that denotes a storage location used to store a data value . A variable may take different values at different times during the execution of the program.

Eg: name, boy

Variable may consists of alphabets, digits, the underscore (_) and dollar character, with following contentions.

1. They must not begin with a digit.
2. Upper case and lowercase are distinct.
3. It should not be a keyword
4. White space is not allowed
5. Variable names can be of any length.

2.1.3 Data Types

Java defines eight simple types of data: byte, short, int, long, char, float, double and Boolean.

These can be put in four groups.

Integers - this group includes byte, short, int and long which are for whole-valued signed numbers

Floating - point numbers: This includes float and double

Characters - this includes char

Boolean - this includes Boolean, which is a special type for representing true/false values

Integers

All the four integer types are signed, positive and negative values. Java doesn't support unsigned, positive-only integers.

The width and the range of the integer types can be specified

Type	width	Range
long	64(8 byte)	-9,233,372,036,854,775,808 to 9,233,372,036,854,775,807
int	32(4 byte)	-2,147,483,648 to 2,147,483,647
short	16(2 byte)	-32,768 to 32,768
byte	8(I byte)	-128 to 127

byte- This is a signed 8-bit type that has a range from –128 to 127. The variables of type byte

are used when you work with a stream of data from a network or file. The variables can be declared as - byte a ;

short - This is a signed 16-bit type that range from -32,768 to 32,768.

int - It is a signed 32 bit type that ranges from -2,147,483,648 to 2,147,483,647

long - long is a signed 64 bit type and is useful when an int type is not large enough to hold the desired value.

Floating –point types

The two kinds of floating point types are float and double which represent single and double precision numbers

Type	**Width**	**Range**
double	64 bits(8 byte)	1.7e-308 to 1.7e+308
long	32 bits(4 byte)	3.4e-038 to 3.4e+038.

float - specifies single precision values. Single precision is faster on some processors and takes half as much space as double precision. float can be useful when representing dollars and cents

double – It uses 64 bits to store a value. double precision is actually faster than single precision. All mathematic, trigonometric functions such as sqrt(), sin(), cos() return double values. double is the best choice if you need to maintain accuracy.

Characters :

The data type used to store characters is **char**. Java uses **Unicode** to represent characters. Unicode defines fully international char set that represents characters of all languages, such as Latin, Greek. Arabic etc. It requires 16 bits. The range of char is 0 to 65536. There are no negative chars.

Ex

```
class Chardemo
{
    public static void main(String args[ ])
   {
      char ch1, ch2;
      ch1 = 88;
      ch2 = “Y”;
```

```
        System.out.println("ch1 and ch2 : " + ch1+" "+ch2);
    }
}
```

The output is : ch1 and ch2 : X Y

where 88 is the ASCII value that corresponds to X.

Booleans :

Boolean is the type used for logical values. It can have any one of the two possible values **true** or **false**. This is the type returned by all relational operators such as a > b. boolean is the type required by conditional expressions that govern the control statements such as if or for.

2.1.4 Declaring Variables:

Variables are the names of storage locations. Its have 3 types.

1. It tells the compiler what the variable name is:

2. It specifies what type of data the variable will hold.

3. The place of declaration decides the scope of the variable.

Variable must be declared before it is used in the program

General Form:

Type variable1, variable2… ....., variable;

Eg: int num;

Giving values:

Variable name= value;

Eg: value=0;

Type Variable= value;

Eg: int total=500;

Reading data from keyboard:

```
Import java.io.DataInputStream;
Class Reading
{
   Public static void main(String args[])
{
   DataInputStream in =new DataInputStream(System.in)
   int intNumber=0;
  float floatnumber=0.0f;
 try
{
        System.out.println("Enter an Integer");
        intNumber=Integer.parseInt(in.readLine());
        System.out.println("Enter a float number:");

        floatNumber=float.valueof(in.readLine()).floatvalue();
}
Catch(Exception e){ }
System.out.println("intNumber="+intNumber);
System.out.println("floatNumber="+floatNumber);
}
}
```

o/p:

```
Enter an integer:
167
Enter a float number:
897.9
intNumber=167
floatNumber=897.9
```

2.1.5 Scope of Variables :

Java variables are classified into 3 types:

- Instant variables
- Class variables
- Local variables

Instance and class variables are declared inside a class.

A block defines the scope. A scope determines what objects are visible to other parts of the program. It determines the lifetime of those objects. In Java the two major scopes are those defined by a class and those defined by a method.

When you declare a variable within a scope, you are localizing the variable and protecting it from unauthorized access and/or modification. Scopes can be nested. Objects declared within inner scope will not be visible outside it.

```
class Scope{
  public static void main(String args[ ]){
     int x;
     x = 12;
      if ( x == 12){
          int y = 6;                  // y  is visible only to this block
          System.out.println("the value of y :" +y);
          x = y + 4;
       }
     y = 5;                  // error as y is not visible here
     System.out.println("the value of x : +x);
   }
}
```

y is defined within the if block. So its value is not visible outside that block. Within the if block x can be used as code within a block has access to the variables declared by an enclosing scope.

Variables are created when their scope is entered, and destroyed when their scope is left. So the variables declared within a method will not hold their values between calls to that method.

2.1.6 Type conversion and casting :

It is common to assign a value of one type to a variable of another type. If the two types are compatible, then Java will perform the conversion automatically. It is possible to assign a int value to a long variable.

Not all types are compatible, so not all type conversions are implicitly allowed. i.e. there is no conversion defined from double to byte. It is possible to get a conversion between incompatible types using the **'casting'**

—

Automatic conversion

Automatic conversion occur under the following conditions

1. The two types are compatible
2. The destination type is larger than the source type

A widening conversion takes place when the conditions are met. Numeric types are compatible with each other. The numeric types are incompatible with char or boolean. char and Boolean are not compatible with each other.

Type casting :

When you assign an int value to a byte variable the conversion will not be performed automatically as a byte is smaller than an int. The conversion is called a **narrowing conversion** as you are explicitly making the value narrower to fit into the target type.

A **cast** is a explicit type conversion. The form is :

(target-type)value

The target-type specifies the desired type to convert the specified value to.

Ex –

```
int  a;
byte b;
 b = (byte) a;
```

Type promotion rules :

1. All bytes and short values are promoted to int
2. if one operand is long, the whole expression is promoted to long
3. If one operand is float, the entire expression is promoted to float.
4. If any of the operands is double, the result is double.

2.2 OPERATORS AND EXPRESSIONS

2.2 Operators :

Operators are characters that instruct the compiler that you wish to perform an operation on some operands. Operators specify operation instructions, while operands are variables, expressions or literal values. Some operators take a single operand these are called unary operators. Some operators come before the operand, these are called prefix operators. And those that come after the operands are called postfix operators. Most operators are between operands and are called as infix binary operators. An operator that takes 3 operands called as ternary operator.

The basic types of operators are –

Arithmetic, Relational, Logical, Assignment, increment & decrement, conditional, Bitwise and Special Operators.

2.2.1 Arithmetic Operators

Arithmetic operators are used to construct mathematical expression as in algebra. Java provides all the basic arithmetic operators. they are :

Operator	Meanings
+	Addition or unary plus
-	Subtraction or Unary minus
*	Multiplication
/	Division
%	Modulo division(Remainder)

Integer Arithmetic : When the both operands in a single arithmetic expression such as a+b are integers, the expression is called an *integer expression* , and the operation is called *integer arithmetic.*

Real Arithmetic : An arithmetic operation involving only real operands is called *real arithmetic.*

Mixed-mode arithmetic : When one operands is real and other is integer, the expression is called *mixed-mode arithmetic.*

Example program for Four function calculator operators

```
class Calculate{
  public static void main(String args[ ]){
      int a = 5;

      int b = a + 2;
      int c = a * b;
      int d = c – b;
      int e = c / d;
      int f = - b;
     System.out.println(“value of b =” +b);
     System.out.println(“value of c =” +c);
     System.out.println(“value of d =” +d);
     System.out.println(“value of e =” +e);
     System.out.println(“value of f =” +f);
  }
}
```

2.2.2 Relational Operators

Java supports six relational operators in all they are :

Operator	Meanings
<	Is less than
<=	Is less than or equal to
>	Is greater than
>=	Is greater than or equal to
= =	Is equal to
!=	Is not equal to

When arithmetic expressions are used on either side of a relational operator, the arithmetic expression will be evaluated first and then the result compared. That is, *arithmetic operators have a higher priority over relational operators.*

2.2.3 Logical Operators :

Java has 3 Logical operators they are,

Operator	Meanings
&&	Logical AND
||	Logical OR
!	Logical NOT

The logical operators && and || are used when we want to form compound conditions by combining two or more relations.

eg : a > b && x == 10

2.2.4 Assignment Operators :

Assignment operators are used to assign the value of an expression to a variable.

2.2.5 Bitwise Logical Operators :

Bitwise NOT

This is the unary NOT operator ^. which inverts all the bits of its operand .

Eg : The number 18 can be represented in binary as 10010 which becomes 01101 after the NOT operator is applied.

Bitwise AND

The AND operator (&) produces 1 when both the operands are 1 , 0 zero is produced in all other cases.

Eg : 1010
& 1110
———
1010

Bitwise OR (|)

In this case , the result is 1 if any one of the operands is 1.

Eg :

1010
| 1001
———
1011

Bitwise XOR (^)

The result is 1 if exactly one operand is 1, else the result is zero.

Eg : 1110

^ 1100

1011

Right shift (>>)

This operator shifts all the bits in a value to the right to a specified number of times. The **syntax is**

value >> num ;

Ex -

```
int a = 16 ;
a = a>>2 ;
```

The result is 4 .

ie the binary value of 16 is 10000 which after right shift becomes 00100 . The decimal value of the result is 4.

The operator shifts the value of 16 to the right two positions, which causes 2 low – order bits to be lost resulting in number being set to 4.

Left shift (<<)

The syntax is value << num ;

Ex

```
int a = 64 ;
a = a << 2 ;
```

The result is 256. ie the binary value of 64 is 0100 0000 is shifted left two positions, and the result is 1 0000 0000 which is 256.

unsigned right shift (>>>)

>> operator automatically fills two high order bits with its previous contents each time a shift occurs. This preserves the sign of the value. If you are shifting something that doesn't represent a

numeric value, you may not want sign extension to take place. In this case you need to shift a zero to the high order bit no matter what the initial value was. This is known as unsigned shift.

2.2.6 Special Operators :

Java supports some special operators of interest such as ***instanceof*** operator and member selection operator ***(.) Dot Operator.***

instanceof operator : The ***instanceof*** is an object reference operator and returns ***true*** if the object on the left-hand sides an instance of the class given on the right-hand side.

Dot Operator : The Dot Opertor (.) is used to access the instance variable and methods of a class objects.

2.2.7 OPERATOR PRECEDENCE

The operators at the higher level of precedence are evaluated first.

Ranks	Operators					
1	. (Dot)	( )	[ ]			
2	-	++	--	!	~	(type)
3	*	/	%			
4	+	-				
5	<<	>>	>>>			
6	<	< =	>	>=		
7	= =	! =				
8	&					
9	^					
10	|					
11	&&					
12	||					
13	? :					
14	=					

2.3 Decision Making and Branching :

2.3 Control or Decision making statements :

Control or Decision making statements are used to alter the flow of execution according the changes to the state of a program.

The control statements are basically categorized into three types.

1. Selection statements
2. Iteration statements
3. Jump statements

2.3.1 Selection statements

These statements make the program to choose different paths based on the result of an expression or the state of a variable. The two selection statements in Java are

1. **if**
2. **switch**

i) Simple if statement :

The **if** statements is used to control the flow of execution of statements. It is a two-way decision statement.

Syntax :

```
if (test expression)
  {
      statements;
  }
      statementX;
```

ii) The If…. Else statement :

The if …. else statement is an extension of the simple if statements.

Syntax :

```
if (test expression)
    {
        True – statement;
    }
Else
    {
        False – statement;
     }
     Statements – X;
```

if the test expression is true, then the true- statements are executed. otherwise the false statements will be executed and in either case , either true-block or false block will be executed, not both. in the both the case the control is transferred to the statements – X.

iii) Nested of If…Then…Else Statements :

When a series of decision are involved, we may have to use more than one if…else statements in nested form.

Syntax:

```
if(test Condition1)
  {
     if(test Condition 2)
       {
          Statements-1;
       }
     else
       {
          Statements-2;
       }
  }
else
  {
     Statements-3;
  }
     Statements – x;
```

Once the first condition is satisfied, we have to move to the inner if statement which checks for the next condition. And depending on its evaluation Statements 1 or 2 is executed.

iv) The Else if Ladder :

There is another way of putting ifs together when multipath decision are involved. A multipath decision is a chain of ifs in which the statements associated with each else is an if.

Syntax :

```
If (condition1)
    Statements-1;
else if (condition2)
    Statements-2;
else if (condition3)
    Statements-3;
    ----------
else if (condition-n)
    Statements-n;
else
    default-Statement;
Statement-x;
```

The condition are evaluated from the top downwards. As soon as true condition is found, the statements associated with it is executed and the control is transferred to the statements-x. when the condition is become false then the final else containing the default-statements will be executed.

v) switch statement

switch is a multi-way branch statement. It provides a better alternative than a if statements. large series of if-else –

Syntax:

.

```
switch(expression) { case value1 :
statement sequence break;
case value2 :
statement sequence break;
case value3 :
statement sequence break;
……. case valueN :
statement sequence break;
default :
statement sequence
}
```

The expression must be of type byte, short, int or char. Each of the values specified in the **case** must be type compatible with the expression. Each value must be a unique literal. Duplicate case values are not allowed.

Nested switch statements

You can use a switch as a part of the statement sequence of an outer switch. This is called a nested switch.

Important notes on switch statement

1. The switch differs from **if** statement in that switch can only test for equality Whereas if can evaluate any type of Boolean expression.
2. No two **case** constants in the same **switch** can have identical values
3. A switch statement is usually more efficient than a set of nested **if**s.

2.4 Decision Making And Looping

2.4.1 Decision Making :

These statements create loops. A loop executes the same set of statements repeatedly until a certain condition is met.

i) while statement

Syntax :

```
while(Condition){
        body of the
loop
```

The condition can be any Boolean expression. The statements within the body of while will be executed till the condition is true. If the condition becomes false, control passes to the statement that follows the loop.

Ex -

```
class Sum
{
    public static void main(String args[ ])
       {
          int i = 1;
          int sum = 0;
          while(i = 10)
            {
              sum = sum + i;
              i = i + 1;
            }
          System.out.println("The sum of first 10 natural numbers :" +sum);
        }
}
```

ii) do – while statement

Sometimes we need to check a condition at the end of the loop rather than the beginning . This can be done with the help of do-while.

Syntax :

```
do{
      body of the loop
   }while(condition);
```

Each iteration of the loop , executes the body of the loop first and then the condition is evaluated. If the condition is true, the loop will repeat else the loop will terminate

```
class Dowhile{
   public static void main(String args [ ]){
      int sum = 0;
      int n = 10;
      int inc = 2;
```

```
    do{
        sum = sum + inc;
        inc = inc + 2;
      }while(inc <=10);
     System.out.println("the sum of even numbers till 10:" +sum);
  }
}
```

iii) for statement

syntax:

```
for(initialization ; condition ; iteration ){
        body of the loop
  }
```

First the initialization part of the loop is executed. Then the condition is evaluated. If the result of the condition or the expression is true, the body of the loop is executed, and the iteration part of the loop is executed. This is repeated till the condition becomes false. If it is false the loop is terminated.

2.5 JUMPS STATEMENTS

2.5.1 Break statement :

Use of break statement

1. To terminate a statement sequence in switch statement
2. To exit a loop
3. As a civilized form of goto.

Using break to exit a loop

By using break, you can force intermediate termination of a loop, bypassing the conditional expression and any remaining code in the body of the loop. When a break statement is encountered inside a loop, the loop is terminated and program control resumes at the next statement following the loop.

```
class Usebreak1{
  public static void main(String args[ ]){
```

```
    for(int i = 1; i<=10;i++){
        if (i  =  =  4) break;
       System.out.println("The value of i is : " + i ) ;
    }
  }}
```

when used inside a set of nested loops, the break statement will only break out of the innermost loop.

```
class Usebreak2{
  public static void main(String args[ ] ){
    for(i  = 1 ; i  <=5 ; i ++ ){

        System.out.println("i  = " +i ) ;
        for( j  = 1 ; j <= 5 ; j ++){
            if ( j = = 3 ) break;
           System.out.println("j = " +j ) ;
        }
     }
  }}
```

Using break as a form of goto

Java doesn't have a goto statement , but it uses break statement as a form of goto.

Syntax :

```
break label1 ;
```

label1 is the name of the label that identifies a block of code. When this statement executes, control is transferred out of the named block of code.

```
class Break{
    public static void main(String args[ ]){
        bolean t = true ;
        first : {
            second : {
                third :{
```

```
                    System.out.println("Before break");
                     if (t) break second ;
                    System.out.println("this wont execute");
                   }
              System.out.println("Within  second");
        }
       System.out.println("After the second block");
     }
     } }
```

2.5.2 Continue statement :

Sometimes we need to continue the execution of the loop, but we need to stop the remainder of the code in the body of the loop. This can be done with the continue statement. It can be used in all the looping statements. In the case of do-while and while statements, continue statement causes control to be transferred directly to the conditional expression that controls the loop. In the case of for loop, continue transfers control to the iteration portion.

```
class Continue{
    public static void main(String args [ ]){
        int i ;
        for(i = 0; i < 10 ; i++){
          System.out.println( i + " ");
          If (i % 2 == 0) continue;
          System.out.println( " ");
        }
    }
}
```

Use of continue in for loop

```
class Continuefor{
   public static void main(String args [ ]){
      outer : for( i = 0 ; i<10 ; i ++){
                  for( j – 0 ; j<10 ; j ++){
                    if (j < i){
```

```
                    System.out.println();
                    continue outer;
                  }
                  System.out.println(" " +(i * j));
                }
            }
                System.out.println();
     }
}
```

2.5.3 Return Statement :

Return is used to explicitly return from a method. It causes program control to transfer back to the caller of the method

```
class Return{
    public static void main(String args [ ]){
            boolean t = true;
            System.out.println("Before return");
            if ( t ) return;
            System.out.println("After checking value of t");
    } }
```

2.6 Classes, Objects and Methods:

2.6.1 Defining Class :

A class is a user defined data type within template the server to define its properties. Once the class type has been defined, we can create "variables" of that type using declaration. In Java these variables are termed as instances of classes, which are the actual *objects.*

```
Class classname [extends superclassname]
 {
    [fields declaration;]
    [method declaration;]
 }
```

2.6.2 Method Declaration :

Methods are declared inside the body of the class but immediately after the declaration of instance variables. The general form of the Method class are :

```
Type methodname (parameter-list)
 {
      Method-body;
 }
```

2.6.3 Creating Objects :

Objects in Java are created using the new operator. The new operator creates an objects of type specified class and returns a referenced to that object.

Eg of creating an object of type Rectangle

```
Rectangle rect1;                  //declare the object
rect1 = new Rectangle ( );  // instantiate the object
```

2.6.4 Constructors :

Constructor have the same name as the class itself. Secondly, they do not specify a return type, not even **Void.** this is because they return the instance of the class itself.

2.6.5 Method Overloading :

In Java, it is possible to create methods that have the same name, but different parameters lists and different definition. This is called ***Method overloading***

Method overloading is used when objects are required to perform similar tasks but using different input parameters.

When we call a method in an object, Java matches up the method name first and then the number and type of parameters to decide which one of the definitions to execute. this process is known as ***polymorphism.***

2.6.6 Static Members :

we have seen that class basically contains two section. One declare variable and the other declares methods. These variable and methods are called ***instance variable and instance methods.***

static int count;

static int max(int x, int y);

The members that are declare static as above is called static members. the static variables and static methods are often referred to as class variables and class methods.

2.6.7 Inheritance : Extending A Class

Reusability is yet another aspect OOP paradigm. Java support this concept.

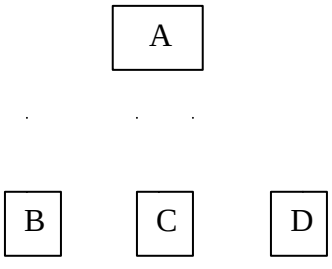
Java classes can be reused in several ways. This is basically done by creating new classes, reusing the properties of existing ones. **The Mechanism of deriving a new class from an old one is called inheritance.**

Inheritances may take different forms :

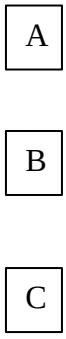
1. Single inheritance (only one super class)
2. Multiple inheritance (Several super class)
3. Hierarchical inheritance (one super class, Many subclasses)
4. Multilevel inheritance (Derived from a derived class)

1. Single inheritance

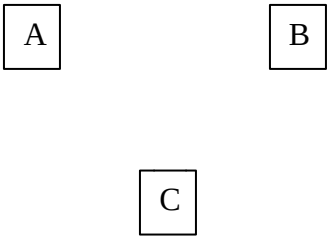
2. Multiple inheritance



3. Hierarchical inheritance



4. Multilevel inheritance



5. 6.8 Over Ridding methods:

A method define in a super class is inherited by its sub class and is used by the objects created by the sub class. Method inheritance enables us to define and use methods repeatedly subclasses with out having to define the methods again in subclass. We want an object to respond to the same method but have different behavior. Override the methods define the super class. The subclass that has the same name, same arguments and same return type as a method in the super class .

Eg:

```
Class super
{
Int x;
Super(int x)
{
 this.x=x;
}
 Void display()                          //method defined
{
   System.out.println("super x=" + x);
}
}
Class sub extends Super
{
 int y;
 sub(int x, int y)
  this.y=y;
}
 Void display( )  {
```

```
System.out.println("super x=" + x);
System.out.println("sub y =" + y);
}
}
Class OverrideTest
{
   Public static void main(String args[])
{
  Sub s1=new sub(100,200);
S1.display();
} }
```

Final variables and methods:

All methods variables can be overridden buy default in subclasses. So we prevent the subclasses from overriding the members of subclasses, so we can declare them as final using the keyword final as a modifier

```
Final int SIZE= 100 ;
Final void shows()
{ ---------- }
```

Here we can easily alter in any way. The value of a final variable can never be changed. final variables, behaves like a class variables and they do not take any space on individual object of the class.

Final class:

We prevent a class being further subclass for security purpose .A class that cannot be sub classed is called a final class. The keyword is final.

Finalizer method:

A constructor method is used to initialize an object when it is declared .this process is known as initialization, java supports a concept called finalization, but it is just opposite to initialization, we know that java run-time is an automatic garbage collecting system. It automatically frees up the memory resources used by the objects. But objects may hold other non-object resources such as file descriptors or window system fonts. The garbage collector cannot

free these resources In order to free these resources we must a finalizer method. This is similar to destructor in c++

Abstract methods and classes:

We can indicate that a method must always be redefined in a subclass, thus making overriding compulsory. Trhis is done using the modifier keyword abstract in the method define.

Eg:

```
Abstract class shape
{
-       - - - - -
-       - - - - - -
Abstract void draw( )
-       - - - -
-       }
```

- While using abstract its have some rules:

1. We cannot use abstract classes to instantiate objects directly. Eg: Shape s=new shape ( ) it's illegal because shape is an abstract class
2. The abstract methods of an abstract class must be defined in its subclass.
3. We cannot declare abstract constructors or abstract static methods.

5.6.9 VISIBILITY CONTROL:

The variables and methods of a class are visible every where ion the program. it may be necessary in some situations to restrict the access to certain variables and methods from ut side the class. Visibility modifiers are pulic, private& protected.

1. Public Access: any variables or method is visible to the entire class in which it is defined . we want to make it visible to all the classes out side this class. This is possible by simply declaring the variable or method as public. A variable or method declared as public has the widest possible visibility abd accessible everywhere.
2. friendly access: the member defaults to a limited version of public accessibility known as friendly level of access. The difference b/w the public access and the friendly access is that the

public modifier makes fields visible in all the classes, regardless of their packages while the friendly access makes fields visible only in the same package.

3. Protected access the visibility level of a protected field lies in b/w the public access and friendly access. That is the protected modifier makes the fields visible not only to all classes and sub classes in the same package but also to subclasses to their package.
4. Private access: private fields enjoy the highest degree of protection. They are accessible only with their own class. They cannot be inherited by subclass and therefore not accessible in subclasses. A method declared as private behaves like a method declared as final. It prevents the method from being sub classed.

Suggested Questions

1. **What are the available data types in Java?**
2. **Explain Branching statements in Java with programs**
3. **Explain Looping statements in Java with programs**
4. **What is known as finalizer method?.**
5. **What is known as final class?.**

UNIT –III

Arrays, Strings and Vectors – Interfaces: Multiple Inheritance –Packages : Putting Classes together – Multithread Programming

---

Array String and vectros

Array is a group of contiguous or related data items that there share a common name. **eg** salary[10] Its hava two types 1.One -dimensional array 2.Two dimensional array

One -dimensional array:

A list of items can be given one variable name using only one subscript and such a variable is called a single variable or a one-dimensional array.Eg: numer[3],x[2]

creating an array: array must declared and creation int he computer memory before they are used.

creation of array involves three steps:

1. declaring array
2. creating memory location
3. putting values in to the memory location.

Declaration of array: Its have two forms

type array name[];	type [] array name;

eg: int number[ ],float[ ] marks

Creation of array: after declaring array we need to create in the memory. java allows us to create array using new operator only.

```
arrayname = new type[size]
```

Eg: number = new int[5]

Initialization of array:

final step is to put values into array created.this process is kbnown as initialization.

```
arrayname[subscript] = value :
```

Eg: number[0]=35;

number[1]=40;

.....

.....

number[4]=19

Java creates array starting with the subscript of) and ends with a value one less than the size specified.

type arrayname[]={list of values}

Array Length: we can obtain the length of the array a using a.length

Eg: intsSize = a. length

```
Class numberSorting
{
public static void main (string args[])
{
int number [ ]={55,40,80,65,71};
int n =number.length; //array length.
System.out.println("Given list:");
for(int i=0;i<n;i++)
{System.out.println(""+number[i]);
}
System.out.println("/n");
//sorting begins:
for(int i=0;i<n;i++)
{
for(int j=i+1;j<n;j++)
{
if(number[i]<number[j])
{
//interchange values
int temp=number[i];
number[i] =number[j];
number[j] =temp;
}
} }
```

```
// sorting end
System.out.println("sortling list")
for(i=0;i<n;i++)
{
System.out.println(""+number[i]);
}
System.out.println(" ")
}
}
```

String :

String represent a sequence of characters. The easiest way to represent a sequence of characters in java by using character array.

Eg:

char charArray[]=new char[4];

String are class objects and implemented using two classes, namely, String and String buffer.A java string is not a character array and is nit NULL terminated. string may be declared and created as follows

```
String string name;
StringName= new String ("String")
```

Eg: String firstName;
firstName = new String("Anil");

String Methods

The String class defines a number of methods that allow us to accomplish a variety of string Manipulation tasks.

Some Most Commonly Used StringMethods

Method Call	***Task performed***
s2 = 51. toLowerCase:	Converts the string sl to all lowercase
s2 = s1. toUppercCase:	Converts the string sl to all Uppercase
s2 = s1. rep 1 ace('x'. *'y');*	Replace all appearances ofx with y
s2 = s1. trim():	Remove white spaces at the beginning and end ofthe string sl
sl.equalsCs2)	Returns 'true' if sl is equal to s2
sl.equalsIgnoreCase(sZ)	Returns 'true' if sl ~ s2, ignoring the case of characters
" sl.length()	Gives the length of sl
sI.ChartAt(n)	Gives nth character of sl
sl.compareTo(s2)	Returns negative if sl < s2, positive if 51 > s2, and zero if 51 is equal s2
sl.concat(s2)	Concatenates sl and s2
II. substrl ng(n)	Gives substring starting from n'th character
sl.substrl ng(n. m)	Gives substring starting from n'th character up to m'" (not including m'h)
Str; ng. Va 1 ueOfCp)	creates a string object of the parameter p (simple type or object)
p. toStri ng()	Creates a string representation of the object p
sl. indexOf(' x')	Gives the position of the first occurrence of 'x' in the string sl
sl. ; ndexOf(, *x'*. n)	Gives the position of 'x' that occurs after nth position in the string s I
Stri ng. Va lueOf (Va ri ab 1 e)	Converts the parameter value to string representation

Alphabetical ordering of strings :

```
class StringOrdering
{
static String name[] ~ {"Madras". "Delhi". "Ahmedabad". "Calcutta", "Bombay"}:
public static void main(String args[ ])
{
for (lnt J=i+1; J < size; j++)
(
```

```
1f (name[J].compareToCname[l]) < 0)
(
/ / swap the strings
Temp= name[i];
name[i]= name[j];
name[j] = temp:
}}
}
for (int 1=0: 1 < size: i++) {
}
System. out. pri nt 1 n C name[ i ] ) ;
}
}
}
```

produces the following sorted list:

Ahmedabad

Bombay

Calcutt

Madras

StringBuffer Class: String Buffer is a peer class of String. While String creates strings of fixed length, String Buffer creates strings of flexible length that can be modified in terms of both length and content. We can insert characters and substrings in the middle of a string, or append another string to the end.

Commonly Used StrlngBuffer Methods

Method	***Task***
sl.selChar'tAt(n. 'x')	Modifies the nth character to x
s1.append' (s2)	Appends the string s2 to sl at the end
sl.insert (n. s2)	Inserts the string s2 at the position n of the string sl

sl.setLength (n)	Sets the length of the string s\ to n. lfn<sl. 1 ength C) [I' n>s 1. length C) zeros are added to S 1

String manipulations:

```
class StrlngManlpulation (
public static void mainCString args[ J) (
StringBuffer  str=~ new Strl ngBufferC'ObJect 1 anguage.);
 System.out.prlntlnC.Orlglnal Strlng :" + str):
/ / obtaining string length
System.out.println(Length of string :. + str.lengthC)):
/ / Accessing characters in a strine
for (int i=0; i < str.1ength( ): i + +)
{
Int P=i+ 1:
System.out.println("Character at position:”+p+”is”+str.CharAt(i));

}
/ / Inserting a string in the middle
String  aString=new String(str.toStrlng());
Int pos= aStclng.indexOf(" language");
str,insert(process," Oriented ");l
System,out,prlntln("Modifled string:”+str);
/ / Modifying characters
Str.setCharAt(6, ‘-’);
System.out.println(“string now:+str”)
/ / Appending a string at the end
Str.append(improves sceurity);
System.out.println(“append String:”+str);
}
}
```

Vector:

The J2SE version supports the concept of variable arguments to methods. This Feature can also be achieved in Java through the use of the Vector class contained in the java.util.package. This class can be used to create a generic dynamic array known as *vector* that can hold *objects of any type* and *any number*. The objects do not have to be homogeneous. Array can be easily implemented as vectors. Vectors are created like array as follows

Vector intVect= new Vector( ): // declaring without size

Vector list = new Vector(3): *II* declaring with size

Note that a vector can be declared without specifying any size explicitly. A vector without size can accommodate an unknown number of items. Even, when a size is specified, this can be overlooked and a different number of items may be put into the vector. Remember, in contrast, an array must always have its size specified. .

Vectors possess a number of advantages over arrays.

1. It is convenient to use vectors to store objects.
2. A vector can be used to store a list of objects that may vary in size.
3. We can add and delete objects from the list as and when .-required. '

A major constraint in using vectors is that we cannot directly stoJe-s1fuple data type in a vector; we can only store objects. Therefore, we need to convert simple types to objects. This can be done using the *wrapper classes* discussed in the next section. The vector class supports a number of methods that can be used to manipulate the vectors created.

The important vector methods:

1ist. addElement(item)	Adds the item specified to the list at the end
list .elementAt(l0)	Gives the name of the 10th object
1ist. size ()	Gives the number of objects present
list. removeElement (item)	Removes the specified item from the list
list . removeElementAt (n)	Removes the item stored in the nth position of the list
list. removeAllElements ()	Removes all the elements in the list
list. copyInto(array)	Copies all items from list to array

list.insertElementAt (item. n) Inserts the item at nth position

Working with vectors and arrays

```
import java.util.*;       // Importing Vector class
class LanguageVector {
public static void main(String args[ ] ) {
vector list=new vector();
int length=args.length;
for(int i=0;i<length;i++)
{1ist. addElement( args [i] ) ;
}
list.lnsertElementAt("COBOL",2);
int size= list.size();
String  listArray[]= new String[size];
1ist. copyInto(1istArray) ;
System.out.pnntln("List of Languages");
For(int i=0; 1 < size; i++) {
System.out.pnntln("listArray[i]");
}
}
}
```

Command line input and output are:

```
C: JAVAlprog>Java LanguageVector List of Languages
Ada
BASIC
COBOL
C+ +
FORTR
AN
Java
```

Wrapper Classes:

Vectors cannot handle primitive data types like int, float, long, char, and double. Primitive data types may be converted into object types by using the wrapper classes contained in the java.lang package.

Wrapper classes for converting simple types

Simple Type	Wrapper class
Boolean	Boolean
Char	Character
Double	Double
Float	Float
Int	Integer
Long	Long

Converting Primitive Numbers to Object Numbers Using Constructor Methods

Constructor Calling	***Conversion Action***
Integer IntVal=new Integer(i);	Primitive integer to Integer object
Float FloatVal=new Float(f);	Primitive float to Float object
Double DoubleVal =new Double(d);	Primitive double to Double object
Long LongVal=new Long(l);	Primitive long to Long object

Converting Object Numbers to Primitive Numbers Using **typeValue( ) method**

Method Calling	***Conversion Action***
lnt i=intVal. lntValue( );	Object to primitive integer
float f = FloatVal.floatValue( );	Object to primitive float
long 1 = LongVal. longValue ( );	Object to primitive long

double d=OoubleVal.doubleValue( ),Object to primitive double

Converting Numbe.rs to Strings Using to String() Method

Method Callinl	*Conversion Action*
Str= Integer.toString(i)	Primitive integer to string
str = Float.toString(f);	Primitive float to string
str ~ DoubletoString(d);	Primitive double to string
str ~ LongtoString(l);	Primitive long to string

Converting String Objects to Numeric Objects Using the Static Method ValueOf( )

Method Calling	*Conversion Action*
*D*oubleVal = Double.Valueof(str);	Converts string to Double obJect
FloatVal=Float.ValueOf(str);	Converts string t0 Float object
IntVal= Inleger.Valueof(str);	Converts string to Integer object
LongVal=Long.ValueOf(str);	Converts string to Long object

Converting numeric strings to primitive numbers using parsing methods

Method Calling	*Conversion Action*
Int i= Integer.parseInt(str)	Converts string in to primitive integer
Long i=Long.parseLong(str)	Converts string in to primitive Long

Enumerated Types

J2SE 5,0 allows us to use the enumerated type in Java using the enum keyword, This keyword can be used similar to the static final constants in the earlier version of Java. For example, consider the following code:

Use* of *enum type data

```
public class Workingdays {
enum Days {
Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday
}
```

```
public static vOld malnCString args[J) {
for ( Days d :Days.values())
{
Weekend(d) ;
}
}
private static void weekend(Days d)
{ if(d. equal s (Days. sunday)
    System.out.println(“value=”+d +”is a holiday”);
else
    System.out.println(“value=”+ d+' is a working day");}}
```

Annotations

The annotations feature, introduced by J2SE 5.0, is also known as metadata. We can use this feature to merge additional Java elements with the programming elements, such as classes, methods, parameters, local variables, packages, and fields.

Metadata is stored in Java class files by the compiler and these class files are used by the JVM or by the program to find the metadata for interacting with the programming elements. Java contains the following standard annotations:

Annotation	***Purpose***
@Deprecat	Compiler warns when deprecated java elements are used in non-deprecated program.
@Overrldes	Compiler generated error when the method uses this annotation type does not override the methods present in the super-class.

In addition to the above standard annotations, Java also contains some meta-annotations available in the java.lang.annotation package. The following table provides the meta-annotations:

Meta-annotation	*Purpose*
@Documented	Indicates annotation of this type to be documented by Javadoc
@Inherited	Indicates that this type is automatically inherited
@ Retention	Indicates the extended period using annotation type
@ Target	Indicates to which program clement the annotation is applicable

The declaration of annotation is similar to that of an interface. We use the symbol '@' before keyword interface. For example, consider the following code that contains the declaration of an annotation:

```
package njunit. annotation:
Import Java. lang. annotat ion. *;
@Retent ion (Retent ionpolicy. RUNTIME)
@Target( ( ElementType . METHOD) )
public @interface UnitTest
{
    Strlng value() :
}
```

where, @Retention is a meta-annotation, which declares that the @UnitTest annotation must be stored in a class file. The @Target meta-annotation is used to declare the @UnitTest annotation, which annotates the methods in the Java class files. The @interface meta-annotation is used to declare the @UnitTest annotation with the member called value, which returns String as an object.

While using annotations, we need to follow some guidelines:

. Do not use extends clause. It automatically extends the marker interface
java.lang.annotation.Annotation.

Do not use any parameter for a method.

Do not use generic methods.

Do not use throws clause

An annotation can also be applied to programming elements. For example, consider the code in which an annotation is applied to the methods, positive() and negative() of a class, Checking:

```
import  nJunit.annotation.*;
public class Checking
 {
   @UnitTest(value=”Test 1. This test will positive"')
   pubilc void positive(int no)
  {
    assert no<0;
  }
 @UnitTest(“Test 2. Th1S test will negative”);
 Public void negative(int no)
{
  assert no<0;}}
```

After merging the annotation with the programming clement, we can use the methods available in the interface, Java.lang.reflect. Annotated Element to query about the existence of programming element and get their values. The methods of the AnnotatedElement interface are:

IsAnnotationPresent()
getAnnotations()
getAnnotation()
getDeclaredAnnotations()

The classes that implement the AnnotatcdElement interface are:

java.lang.retlecLacecssibleobject
java.lang.class
java.lang.reflect.constructor
java.lang.reflect .field
java.lang.refelect.method
java.lang.package

Use* of *annotations

```
lmport Java.lang.annotation.*;
import Java.lang.reflect.*;
 @Retention(RetentionPolicy. RUNTIME)
@lnterface MySingle
{
int value(): // this variable name must be value
}
public class Single
{

// Annotate a method using a marker.
@MySingle(lOO)
Public static void  myMeth()
{
Single ob=~ new Single():
 try
{
Method m= ob.getCl ass (). getMethod( "myMeth");
 MySingle anno= m.getAnnatation(MySingle.class);
 System.out.println("The value is : "anna.value”);      // displays 1 00
}
catch ( NoSuchMethodExcept ion exc)
{
System.out.println("Method Not Found, "):
}
public static void main(String args[]) {
myMeth ( ) :
}
}
```

The output of the above program is The value is: 100

Interfaces: Multiple Inheritances

Introduction

We discussed about classes and how they can be inherited by other classes. we also learned about various forms of inheritance and pointed out that Java does not support multiple inheritance. That is, classes in java cannot have more than one superclass. For instance ,a definition like

```
Class A extends B extends C
{
……
……
}
```

Is not permitted in Java. However, the designers of Java could not overlook the importance of multiple inheritance. A large number of real-life applications require the use of multiple inheritance whereby

We inherit methods and properties from several, distinct classes. since c++ like implementation of multiple inheritance proves difficult and adds complexity to the language, Java provides an alternate approach known as interfaces to support the concept of multiple inheritance.

Defining Interfaces

An interface is basically a kind of class. Like classes, interfaces contain methods and variables but with a major difference. The difference is that interfaces define only abstract methods and final fields. This means that interfaces do not specify any code to implement these methods and data fields contain only constants. Therefore, it is the responsibility of the class that implements an interface to define the code for implementation of these methods.

The syntax for defining an interface is very similar to that for defining a class. The general form of an interface definition is:

```
Interface InterfaceName
{
  Variable declaration;
  Methods declaration;
}
```

Here, interface is the keyword and InterfaceName is any valid Java variable(just like class names)
.

Variables are declared as follows:

```
Static final type VariableName=value;
```

Note that all variables are declared as constants. Methods declaration will contain only a list of methods without any body statements. Examples:

```
return-type
methodName1(parameter_list);
```

Here is an example of an interface definition that contains two variables and one method;

```
Interface Item
{
static final int code=1001;
static final String name=”Fan”;
void display();
}
```

Note that the code for the method is not included in the interface and the method declaration simply ends with a semicolon. The class that implements this interface must define the code for the method.

Another example of an interface is:

```
interface Area
{
final static float pi=3.142F;
float compute(float x,float y);
void show();
}
```

Extending Interfaces

Like classes, interfaces can also be extended. That is, an interface can be subinterfaced from other interfaces. The new subinterface will inherit all the members of the superinterface in the manner similar to subclasses.This is achieved using the keyword extends as shown below:

```
interface name2 extends name1
{
    body of name1
}
```

For example, we can put all the constants in one interface and the methods in the other. This will enable us to use the constants in classes where the methods are not required. Example:

```
interface ItemConstants
{
  int code=1001;
string name=”Fan”;
}
interface Item extends ItemConstants
{
void display();
}
```

The interface Item would inherit both the constants code and name into it. Note that the variables name and code are declared like simple variables. It is allowed because all the variables in an interface are treated as constants although the keywords final and static are not present.

We can also combine several interfaces together into a single interface. Following declarations are valid:

```
interface ItemConstants
{
  int code=1001;
  String name=”Fan”;
}
interface ItemMethods
{
void display();
}
interface Item extends ItemConstants,ItemMethods
{
      ……..
      ……..
}
```

While interfaces are allowed to extend to other interfaces subinterfaces cannot define the methods declared in the superinterfaces. After all, subinterfaces are still interfaces, not classes. Instead, it is the responsibility of any class that implements the derived interface to define all the methods. Note that when an interface extends two or more interfaces, they are separated by commas.

It is important two remember that an interface cannot extend classes. This would violate the rule that an interface can have only abstract methods and constants.

Implementing Interfaces :

Interfaces are used as “superclasses” whose properties are inherited by classes. It is therefore necessary to create a class that inherits the given interface. This is done as follows:

```
class  classname  extends  superclass
```

```
implements interface1,interface2,…

{

  body of classname

}
```

This shows that a class can extend another class while implementing interfaces.
When a class implements more than one interface, they are separated by a comma. The implementation of interfaces can take various forms as illustrated.

Implementation of interfaces as class types is illustrated by the following program. In this program ,first we create an interface Area and implement the same in two different classes, Rectangle and Circle. We create an instance of each class using the new operator. Then we declare an object of type Area, the interface class. Note ,we assign the reference to the Rectangle object rect to area. When we call the compute method of area, the compute method of Rectangle class is invoked. We repeat the same thing with the Circle object.

Program:

```
//Interface Test.java
interface Area // Interface defined
{
        final static float pi=3.14F;
        float compute (float x,float y);
}
class Rectangle implements Area
{
public float compute (float x,float y)
{
        return(x*y);
}
}
```

```
class Circle implements area
{
public float compute(float x,float y)
{
        return(pi*x*x);
}
}
class InterfaceTest
{
public  static void main(String args[])
{
Rectangle rect=new Rectangle();
Circle cir=new Circle(0;
Area area;
area=rect;
System.out.println("Area of rectangle ="+area.computet(10,20));
area=cir;
System.out.println("Area of Circle="+area.computet(10, 0));
}
}
```

The output is as follows

Area of Rectangle=200

Area of Circle=314

Any number of dissimilar classes can implement an interface. However ,to implement the methods, we need to refer to the class objects as types of the interface rather than types of their respective classes. Note that if a class that implements an interface does not implement all the methods of the interface, then the class becomes an abstract class and cannot be instantiated.

Accessing Interface Variables :

Interfaces can be used to declare a set of constants that can be used in different classes. This is similar to create header files in c++ to contain a large number of constants. since such interfaces do not contain methods, there is no need to worry about implementing any methods. The constant values will be available to any class that implements the interface. The values can be used in any method, as part of any variable declaration, or anywhere we can use a final value .

Example:

```
interface A
{
        int m=10;
        int n=50;
}
class B implements  A
{
        int x=m;
void method(int size)
{
…….
…….
if(size<n)
……..
}
}
```

Program2: Implementing multiple inheritance

```
class Student
{
int rollNumber;
void getNumber(int n)
{
        rollNumber=n;
}
```

```
void putNumber(0
{
        System.out.println("Roll No :"+rollNumber);
} }
class Test extends Student
{
float part1,part2;
void getMarks(float m1,float m2)  {
part1=m1;
part2=m2;
}
void putMarks()
{
System.out.println("Marks obtained");
System.out.println("Part 1="+part1);
System.out.println("Part2="+part2);
} }
interface Sports
{
float spotwt=6.0F;
void putwt();
}
class Results extends Test implements Sports
{
float total;
public void putwt(0
{
System.out.println("Sports wt ="+sportswt);
}
void display()
{
```

```
total=part1+part2+sportwt;
putNumber();
putMarks();
putWt(0;
System.out.println("Total score ="+total);
} }
class Hybrid
{
public static void main(String args[])
{
Results student1=new Results();
student1.getNumber(1234);
student1.getMarks(27.5F,33.0F);
student1.display():
} }
```

Output of the program is

Roll No:1234
Marks obtained
Part1=2705
Part2=33
Sports wt =6
Total score=66.5

Packages : Putting classes together

Packages are containers for classes that are used to keep the class namespace compartmentalized. Packages are stored in a hierarchical manner and are explicitly imported into a new class definition

A Java source file may contain any one or all of the following four internal parts

1. A single package statement
2. Any number of import statements
3. A single public class declaration
4. Any number of classes private to the package

Consider a programmer has given a name “Sample” for the class he defines. If you define a class with the same name “Sample” a name collision occurs. You can avoid this name collision by defining your class Sample within a different package. So you can access the class within your package by specifying Packagename**.**Sample. So the class inside your package is not accessible outside your package. Only the class members in the same package can access the class.

DEFINING A PACKAGE

To create a package, include package command as the first statement in your program followed by the package name.

Package pkg **;**

Where pkg is the name of the package.

Any classes inside the file belong to the specified package. Java uses file system directories to store packages. i. e if you have a package say

package Mypackage ;

All the **.** class files for the classes you describe in Mypackage must be stored in a directory called Mypackage. There can be any number of files with the same package statement. You can also create a hierarchy of packages i.e

package pkg1[**.**pkg2] [**.** pkg3] **;**

We can’t rename a package without renaming the directory in which the classes are stored.

CLASSPATH

Consider that you create a class called Sinterest in a package called Myspace. Since the directory name has to match with your package, create a directory by name Myspace and put Sinterest.java in that directory and compile the file. The class file Sinterest.class will be in the directory in the Myspace directory.

To make this work, you have two ways

1. You can go one hierarchy up by one level and try it as java Myspace**.**Sinterest or
2. You can set the Classpath as

   ; c:\Myspace; c:\java\classes

Ex-

```
Package Myspace;
public class Interest{
   int n;
   float p ;
   float v;
   public Interest(int n, float p, float r){
        this.n = n;
        this.p = p;
        this.r  = r;
   }
   public void calculate(){
       System.out.println("Interest"+(n*p*r));
   }
}
```

IMPORTING PACKAGES

If you want to access a member of a class in a package, you need to specify the packagename**.**classname. Sometimes if a class belongs to a multilevel hierarchical package you

need to type all the packages for every class you need to use. You can avoid this by using **import** statement.

In the java source file, the import statement occur immediately following the package statements and before any class definitions.

The general form is :

import pkg1.pkg2.pkg3.(classname/*);

You can either specify a explicit classname or a star which indicates that the system should import the entire package. For ex, we will use our Class Sinterest which belongs to Myspace package.

```
import Myspace ;
public class Intdemo{
   public static void main(String args[ ]){
     Interest i = new Interest(4, 500, 2);
     i.calculate();
  }
}
```

This example accesses the members of the class Interest in the package Myspace. So the package has to be imported before its class members are used.

Multithreaded programming:

Those who are familiar with the modern operating systems such as Windows 95 and windows xp may recognize that they can execute several programs simultaneously, that is called multitasking. In system terminology multithreading. Thread is similar to a program that has a single flow of control. it has beginning , a body ,and an end, and executes commands sequentially. A program that contains multiple flows of control is known as multithreading

The ability of a language to support multi thread is referred to as concurrency.

It is important to remember that threads running in parallel do not really mean that they actually run at the same time.

Multithreading is a powerful programming tool that makes java distinctly different from its fellow programming languages. It is useful in a number of ways. It enables programmers to do multiple things as one time.

Creating thread:

Threads are implemented in the form of objects that contain a method call run(). It is the heart and soul of any thread. It makes up the entire body of a thread and is the only method in which the thread behavior can be implemented.

```
Public void run()
{
…….
……        (statements for implementing thread)
…….
}
```

The run ( ) thread should be invoked by an object of the concerned thread. This can be achieved by creating the thread and initiating it with the help of another thread method called strat( ).

A new thread can be created in two ways:

1. by creating thread class: define a class that extends thread classs and override is run () method with the code required by the thread.

2. by converting a class to a thread: Define a class that implements runnable interface. The runnable interface has only one method. Run() , that is to be defined in the method with the code to be executed by the thread.

Extending the thread class:

The class runnable as thread by extending the class java.lang.thread. This gives us access to all the thread methods directly.

1. Declare the class as extending the thread class.

2 Implement the run() method that is responsible for executing the sequence of code that the thread will execute.

3. Create a thread objects and call the start( ) method to initiate the thread execution.

Declaring the thread:

The thread class can be extended as follows:

```
Class MyThread extends Thread
{
---------
-------
}
```

Now we have new type of thread MyThread

Implementing the run() method

The run() method has been inherited by the class MyThread. We have to override this method in order to implement the code to be executed by our thread. The basic implementation of run()

```
Public void run( )
{
----------          // Thread code here
-----
}
```

When we start the new thread.java calls the thread's run( ) method, so it is the run( ) where all the action takes places.

Starting new thread :

We must follow this code to start the new thread.

```
MyThread a Thread =new MyThread( );
aThread.start( );     //invokes run( ) method
```

The first line instantiates a new object of class MyThread. This statement creates the object. The thread will run the object is not yet running. The thread is in a newborn state. The second line calls the start( ) ,method causing the thread to movein to the runnable state. Then the

java runtime will schedule the thread to run by invoking is run( ) method, The thread is said to be in the running state.

//program

Creating threads using the thread class:

```
//**********************
classA extends thread
{
Public void run( )
{ for ( int i=1;i<=5;i++)
{ Sytem.out.println("\t From ThreadA:i="+i"");
}
System.out.println("Exit form A");
}
}
ClassB extends Thread
{
Public void run( )
{
For(int j=1;j<=5;j++)
{
System.out.println("\t Form Thread B:j="+j);
}System.out.println("Exit from B");
}
ClassC extends Thread
{
```

```
Public void run( )
{
For(int k=1;k<=5;k++)
{
System.out.println("\t Form Thread C : K="+k);
}System.out.println("Exit from c");
}
Class threadTest
{
Public static void main(Str5ing agrs[])
{
 new A( ).start( );
 new B( ).start( );
 new C( ).start( );
}
}
```

Stopping and blocking a thread:

Stopping a thread:

When we want to stop a thread, we calling its stop( ) method, like a thread.stop( ); This statement causes the thread to move to the dead state. A thread will also move to the dead state automatically when it reaches the end of its method. The stop( ) method may be used when the premature death of a thread is desired.

Blocking a thread

A thread can also be temporarily or blocked from entering in to the runnable and subsequently running state by using either of the following thread methods.

sleep( ) //blocked for a specified time.

suspend ( ) // blocked until further orders

wait () // blocked until certain condition occurs

These methods cause the thread to go into the blocked (or not-runnable state). the thread will return to the runnable state when the specified time is elapsed in the case of sleep(), the resume() method is invoked in the case of suspend(),and notify() method is called in the case of wait().

Life cycle of a thread ():

1. new born state
2. Runnable state
3. Running state
4. Blocked state
5. Dead state.

New born state:

when we create the thread object, the thread is born and is said to be in new born state. The thread is not yet scheduled of running. At the state, we can do only one following step.

*Schedule it for running using start () method.

* Kill it using stop () method.

Runnable state:

The runnable state means that the thread is ready for execution and is waiting for the availability of the processor. That is, the thread has joined the queue of threads that are waiting for execution. If all threads have equal priority, then they are given time slots for execution in round robin technique i.e., first come first serve manner. the thread that relinquishes control joins the queue at the end and again waits for its turn. This process of assigning time to threads is known as time-slicing.

If we want a thread to relinquish control to another thread to equal priority before its turn comes, we can do using so by using yield() method

Running State:

Running state means that the processor has given its time to the thread for its execution .the thread runs until it relinquishes control on its own or it is preempted by a higher priority thread. A running thread may relinquish its control in one of the following situations:

1. It has been suspended using in one suspend ( ) method. A suspended thread can be revived by using the resume ( ) method. This approach is useful when we want to suspend a thread for some time due to certain reason, but do not want kill it

2. It has been made to sleep. We scan put the thread to sleep for a specified time period using the method sleep (time) where time is in milliseconds. this means that the thread is

out of the queue during this time period. The thread re-enters the runnable state as soon as this time period is elapsed.

3. It has been told to wait until some event occurs. This is done using the wait ( ) method. The thread can be scheduled to run-again using notify ( ) method

Blocked state:

A thread said to be blocked when it is prevented from entering into the runnable state and subsequently the running state. This happens when the thread is suspend, sleeping, or waiting in order to satisfy certain reqirements.A blocked thread is considered "not runnable" but not dead and therefore fully qualified to run again.

Dead State:

Every thread has a life cycle.A running thread ends its life when it has completed executing its run ( ) method. It is a natural death. A thread can be killed as soon it is born. Or while it is running, or even when it is in "not runnable"(blocked) condition.

Using thread methods:

Thread class methods can be used to control the behavior of the thread. We have used the methods start ( ) & run ( )., There are also methods that can move a thread from one state to another , like yield( ), sleep( ), stop ( ) methods.

```
// program:
Class A extends Thread
{   public void run( )
  {    for( int i=1;i<=5;i++)
     { if(i= =1) yield( );
System.out.println("\t From thread A: 1=" +i);
}
System.out.println("Exit form A");
}
}
Class B extends Thread
```

```
{
Public void run( )
{
For(int j=1;j<=5;j++)
{
System.out.println("\t Form Thread B:j="+j);
If( j= =3) stop( );

}System.out.println("Exit from B");
}
}
Class C extends Thread
{
Public void run( )
{
For(int k=1;k<=5;k++)
{
System.out.println("\t Form Thread C : K="+k);
If ( k= =1)
 Try
{
Sleep (1000);
}
Catch (Exception e)
{
}
}System.out.println("Exit from c");
}
Class threadMethods
{
Public static void main(String agrs[])
```

```
{
 A threadA = new A( );
B threadB = new B( );
C threadC = new C( );
System.out.println("Start thread A");
ThreadA.start( );
System.out.println("Start thread B");
ThreadB.start( );
System.out.println("Start thread C");
ThreadC.start( )
}
}
```

Thread Exception:

Call to sleep ( ) method is enclosed in a try block and followed by a catch block. This is necessary because the sleep ( ) method throws an exception, which should be caught if be fail to catch the exception, program wil not compile.

Java runs system will throw **Illegalthreadstateexception** when ever we attempt to invoke a method that a thread cannot handle in the given state. For eg: a sleeping thread cannot deal with the resume () method. Because a sleeping thread cannot receive any instructions. The same is true with the suspend ( ) method, when it's used an a blocked (not runnable) thread. When ever we call a thread method that is likely to throw an exception, we have to supply a appropriate exception handler to catch it. The catch statement may take one of the following forms.

```
Catch(ThreadDeath e)
{
-            - - - -
- ---------- // killed thread
}
Catch(InterruptedExecption  e)
```

```
    {
-           - - - -
- ---------- //cannot handle it in the current state
-           }
    Catch(IllegalArgumentExecption  e)
    {
-           - - - -
- ---------- //Illegal method argument
-           }
    Catch(Execption  e)
    {
-           - - - -
- ---------- //any other
-           }
```

Thread priority:

In java, each thread is assigned a priority, which affects the order in which it is scheduled for running. The threads of the same priority are given equal treatment by the java scheduler and, therefore they share thre processor on a first-come first serve basis. Java permits us to set the priority of a thread using the SetPriority()method:

```
ThreadName.setpriority(int  Number);
```

The intNumber is an integer value to which the threads priority is set. The thread class defines several priorities Constance.

MIN_PRIORITY = 1

NORMAL_PRIORITY = 5

MAX_PRIORITY = 10

The intNumber may assume one of these Constance or any value b/w 1&10, the default setting is normal priority**.** Most user –level processors should use Norm_priority ,plus or minus1. Background task such as network I/O and screen repainting should use a value very near

to the lower limit. We should be very cautious when trying to use very high prioruity values. These may defeat the very purpose of using multi thread.

By assigning priorities to threads, we can ensure that they are given the attention (or lack of it) deserved. For eg: we may need to answer an input as quickly as possible. When ever multiple threads are ready for execution, the java system chooses the highest priority thread and executes it. For a thread of lower priority to gain control, one for the following things should happen.

1. It stops running at the end of run ( ).
2. It is made to sleep using sleep ( )
3. It is told to wait using wait ( ).

If another thread of a higher priority comes along the currently running thread will be preempted by the incoming thread thus forcing the current thread to move to the runnable state. Remember that the highest priority always preempts any lower priority threads.

```
//Use of priority threads://
Class A extends Thread
{   public void run( )
   {    for( int i=1;i<=5;i++)
       { if(i= =1) yield( );
System.out.println("\t From thread A: 1=" +i);
}
System.out.println("Exit form A");
}
}
Class B extends Thread
{
Public void run( )
{
System.out.println("Thread B started");
For(int j=1;j<=4;j++)
```

```
{
System.out.println("\t Form Thread B:j="+j);
}System.out.println("Exit from B");
}
}
Class C extends Thread
{
Public void run( )
{               System.out.println("Thread C started");
For(int k=1;k<=4;k++)
{
System.out.println("\t Form Thread C : K="+k);
}System.out.println("Exit from c");
}
Class ThreadPriority
{
Public static void main(String agrs[])
{
 A threadA = new A( );
B threadB = new B( );
C threadC = new C( );
threadC.setPriority(Thread.MAX PRIORITY);
threadB.setPriority(ThreadA.getPriority()+1);
threadA.setPriority(Thread.MIN PRIORITY);
System.out.println("Start thread A");
ThreadA.start( );
System.out.println("Start thread B");
ThreadB.start( );
System.out.println("Start thread C");
ThreadC.start( )
System.out.println("End of the main thread");
```

}
}

Synchronization:

we have seen threads that use their on data and methods provided inside their run( )methods, what happens, when they try to use data and methods out side themselves? On such occasions, they may complete for the same resource and may lead to serious problems for eg: one thread may try to read a record form a file while another is still waiting to the same file. depending on the situation we may get strange results. Java enables us to over come this problem using a technique is known as synchronization

Incase of java, the keyword synchronized helps to solve such problems by keeping a watch on such locations. For eg: the method that will read information from a file and method that will update the same file may be declared as synchronized.

```
Synchronized void update ()
{
.....// code here is synchronized
. . . . . }
```

When we declared a method synchronized, java creates a monitor and hands it over to the thread that calls the method first time. As long as the thread holds the monitor no toher the thread can enter the synchronized section of code. A monitor is like a key and the thread that holds the key can only open the lock.

```
Synchronized (lock – object)
{  . . . . . .
. . . . . // code here is synchronized
```

When ever the thread has completed its work of using synchronized method (or block of code), it will hand over the monitor to the next thread that is ready to use the same resource.

Two or more threads are waiting to gain control of a resource. due to some reasons, the condition on which the waiting threads rely on to gain control does not happen .the result in what is known as deadlock.

Implementing the Runable Interface:

It's the one way of creating the thread. That is implementing the runnable interface the thread class is used for creating and running threads, in this section we are going to see how to make the use of Runnable interface to implement the threads.

The Runnable interface declares the run( ) method that is required for implementing threads in our program. Its have steps these are:

1. Declare the class as implementing the runnable interface.
2. Implement the run ( ) method.
3. Create a thread by defining an object that is instantiated from this "runnable" class as the target of the thread.
4. Call the thread start( ) method to run the thread.

```
Class X implements Runnable
{
Public void run()
{
For(int i=1; I<=10;i++)
{  System.out.println("\t Thread X:" +1 );
}
System.out.println("End of Thread X );
 }  }
Class RunnableTest
{                    public static void main (String args[])
{                     X runnable = new X ();
Thread thread = new X();
Thread thread = new Thread (runnable);
System.out.println("End of the thread");
}                    }
```

O/P :-End of the thread

ThreadX =1 ThreadX =2 ThreadX =3 ThreadX =4 ThreadX =5 ThreadX =6

ThreadX =7 ThreadX =8 ThreadX =9 ThreadX =10

Suggested Questions :

1. **What are the methods in string class?**
2. **Explain multiple Inheritance**
3. **Explain packages with program**
4. **Discuss about the Multithread.**
5. **What is known as vector.**

UNIT- IV

Managing Errors and Exceptions – Applet Programming – Graphics Programming

---

4.1 MANAGING ERRORS AND EXCEPTION:

Errors are the wrong that can make a program go wrong.

4.1.1 TYPES OF ERROR :

1. **Compile time error**: All syntax errors will be detected and displayed by the java compiler and therefore these error are known as compile-time error. Whenever the error displays an error, it will not create the .class file.

The most common errors are:

* Missing Semicolons
* Missing (or mismatch of) brackets in classes and methods
* Misspelling of identifiers and keyword
* Missing double quotes in strings
* Use of un declared variables
* Incompatible types in assignments/initialization
* Bad references to objects
* Use of = in place of ==operator.

2. **Run-time error:** Some times, a program may compile successfully creating the .class file but may not run properly. Such programs are may produce wrong results due to wrong logic or may terminate due to errors such as stack overflow.

Most common run time errors are:

* Dividing an integer by zero.
* Accessing an element that is out of the bounds of a array
* Trying to store a value in o an array of an incompatible class or type
* Trying to cast an instance of a class to one of its subclasses
* Passing a parameter that is not in valid range or vale for a method

* Trying to illegally change the state of a thread.

* Attempting to use a negative size for an array

* Using a null object references as a legitimate object references to access a method or a variable

* Converting invalid string to a number.

* Accessing a character that is out of bounds of a string.

```
Class Error2
{
    Public static void main (String args[])
     {
       int a=10,b=5,c=5;
       int x=a / (a-b)  // Division by Zero
       System.out.println("x="+x);
       int y= a/(b+c);
       System.out.println("y="+y);
     }
}
```

4.1.2 EXCEPTIONS:

An Exception that is caused by a run-time error in the program. when java interpreter an counters an error such as dividing by zero. It creates an exception object and throws it. If the exception object is not caught properly the interpreter will display an error message and it will terminate the program.

If we want the program to continue with the execution of the remaining code. Then we should try to catch the exception object thrown by the error condition and then display an appropriate message for talking corrective actions. This is known as **exception handling.**

The purpose of exception handling mechanism is to provide a means to detect and report an "exceptional circumstance" so that appropriate action can be taken. The mechanism suggests incorporation of a separate error handling code that performs the following tasks:

1. Find the problem,(Hit the exception)

2. Inform that an error has occurred (Throw the exception)
3. Receive the error information (catch the exception)
4. Take corrective actions (Handle the exception)

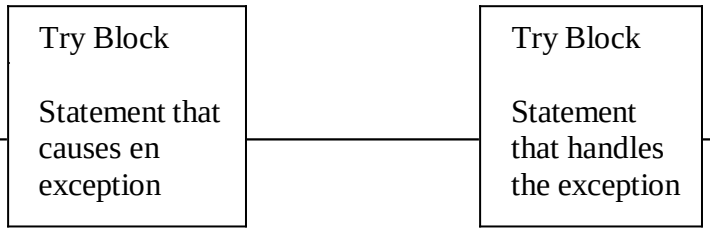
The error handling code basically consists of two segments, one to detect errors and to throw exceptions and the other to catch exceptions to take appropriate actions.

Common Java Exceptions:

Exception Type	**Cause of exception**
ArithmeticException	Caused by math error such as division by zero
ArrayIndexOutOfBoundsException	Caused by bad array indexes
ArrayStoreExceptiop	Caused when a program tries to store the wrong type of data in an array.
FileNotFoundException	Caused by an attempt to access a non existent file
IOException	Caused by general I/O failures, such as inability to read from a file.
NullPointerException	Caused by referencing a null object.
NuymberFormatException	Caused when a conversion b/w strings and number fails.
OutOfMemoryException	Caused when there's not enough memory to allocate a new object.
	Caused when an applet tries to perform an action not allowed by the browsers security setting.
StackOverFlowException	Caused when the system runs out of stack space.
Index Out f BoundsException	Caused when a program attempts to access a nonexistent character position in a string

4.1.3 SYNTAX OF EXCEPTION HANDLING CODE:

The basic concepts of exception handling are throwing an exception and catching it.



Exception object creator	throws exception object	Ecxeption handler

Java uses a keyword try to preface a block of code that is likely to cause an error condition and "throw" an exception. A catch block defined by the keyword catch "catches" the exception "thrown" by three try block and handles it appropriately. The catch block is added immediately after the try block.

```
……..
……..
 try
{   statement;       // generates an exception
}
Catch (Exception – type   e)
{statement ;      //process the exception
}
…….
…….
```

The try block can have one or more statements that could generate an exception. If anyone statement generates an exception, the remaining statement in the block are skipped and execution jumps to the catch block that is placed next to the try block.

The catch block too can have one or more statements that are necessary to process the exception. Remember that every try statement should be followed by at least one catch statement, otherwise compilation error will occur.

The catch statement works like a method definition. The catch is passed a single parameter, which is reference to the exception object thrown (by the try block).If the catch parameter matches with the type of exception object, then the exception is caught and statements in the catch block will be executed , otherwise , the exception is not caught and default exception handler will cause the execution to terminate.

```
// usibg try and catch for exception handling
Class Error3
{    public static void main( String args[] )
  {    inta=10, b=5, c=5, x,y;
 try
{ x =a / ( b-c )         //Exception here
}
Catch ( ArthmeticException  e)
{    System.out.println("Division by zero");
}
Y = a / (b+c);
System.out.println("y="+y);     }      }
```

O/P: Division by zero

Y =1

```
//catching invalid command line arguments:
Class CLineInput
{   public static void main(Stirng args[])
{    int invalid = 0;       //number of invalid arguments
    Int number,count = 0;
   for( int i=0; i< args. Length; i++)
{ try   {  number =Integer.parseInt(args [i]);
}
Catch(NumberFormatException   e)
{     invalid =invalid +1;             //'caught an invalid number
    System.out.println("Invalid Number:" + args[i]);
Continue;
}
Count =count +1;
}
System.out.println("valid Numbers=" + count);
```

```
System.out.println("Invalid  Numbers="+invalid);
}    }
```

4.1.4 MULTIPLE CATCH STATEMENTS:

```
……..
……..
 try
{   statement;       // generates an exception
}
Catch (Exception – type1  e)
{statement ;      //process the exception type 1
}
Catch (Exception – type2  e)
{statement ;      //process the exception type 2
}
Catch (Exception – type3  e)
{statement ;      //process the exception type 3
}
…….
…….
```

Using multiple catch blocks:

```
Class  Erro4
 {
   public static void main (String args[])
   int a[]={5,10};
   int b=5;
   try
```

```
    {
       int x=a[2] /bla[1];
    } catch (ArithmeticException e)
    {
      System.out.println("Division by zero");
    } catch (ArrayIndexOutOfBoundsException e)
    {
      System.out.println("Array Index Error");
    } catch (ArrayStroeException e)
    {
     System.out.println("Wrong data type");
    }
      int y=a[1]/ a[0];
      System.out.println("y="+y);
    }
}
```

4.1.5 USING FINALLY STATEMENT:

Java supports statement known as finally statement can be used to handle exception that is not caught by any of the previous catch statement, finally block can be used to handle any exception generated within a try block. It may be added immediately after the try block or after the last c

try {
- - - - -
- - - - }
finally {
- - - -
- - - - }
	try {
	- - - - -
	- - - -}
	Catch (.....)
	{ - - - - -
	- - - -}

	finally { - - - - - - - - }

4.1.6 THROWING OUR OWN EXCEPTION :

There may be times when we would like to throw our own exceptions. We can do this by using the keyword **throws** as follows.

```
throw new Throwable_subclass;
```

Eg: throw new ArithmeticException( );
throw new NumberFormatExecption( );

4.1.7 USING EXCEPTIONS FOR DEBUGGING :

The exception handling mechanism can be used to hide errors from rest of the program. It is possible that programmers may misuse this technique for hiding errors rather than debugging the code. Exception handling mechanism may be effectively used to locate the type and place of errors.

4.2 APPLET PROGRAMMING

4.2.1 INTRODUCTION :

Applets are small java programs that are primarily used in Internet computing, they can be transported over the Internet from one computer to another and run using the Applet Viewer or any Web browser that supports Java. An applet, like any application program, can do many things for us.

It can perform arithmetic operations, display graphics, play sounds, accept user input, create animation, and play interactive games.

Java has revolutionized the way the Internet users retrieve and use documents on the world wide network. Java has enabled them to create and use fully interactive multimedia Web documents.

A webpage can now contain not only a simple text or a static image but also a Java applet which, when run, can produce graphics, sounds and moving images. Java applets therefore have begun to make a significant impact on the World Wide Web.

Local and Remote Applets :

We can embed applets into web pages in two ways. One we write our own applets and embed them into web pages. Second; we can download an applet from a remote computer system and then embed it into a web page.

An applet developed locally and stored in a local system is known as a local applet. When a web page is trying to find a local applet, it does not need to use the internet and therefore the local system does not require the internet connection. It simply searches the directories in the local system and locates and loads the specified applet.

A remote applet is that which is developed by someone else and stored on a remote computer connected to the Internet. If our system is connected to the Internet, we can download the remote applet onto our system via at the Internet and run it.

In order to locate and load a remote applet, we must know the applet's address on the web. This address is known as Uniform Resource Locator (URL) and must be specified in the applet's HTML document as the value of the CODEBASE attribute.

CODEBASE = http: //www.netserve.com/applets

In the case of local applets, CODEBASE may be absent or may specify a local directory.

4.2.2 How Applets Differ from Applications ?

Although both the applets and stand-alone applications are Java programs, there are significant differences between them. Applets are not full-featured application programs. They are usually written to a accomplish a small task or a component of a task. Since they are usually designed for use on the Internet, hay imposes certain Limitations and restrictions in their design.

How differs from Application:

*.Applet do not use the main ( ) method for initiating the execution of the code. Applets, when loaded, automatically call certain methods of Applet class to start and execute the applet code.

* unlike stand-alone applications, applets cannot be run independently. They are run from inside a web page using special feature known as HTML.

* Applets cannot read from or write to the files in the local computer.

* Applets cannot communicate with other servers on the work.

* Applets cannot run any program from the local computer.

* Applets cannot are restricted from using through libraries from other language such as C or C++.

4.2.3 PREPARING TO WRITE APPLETS:

we have been creating simple java applications programs with a single main ( ) method that created objects, set instance variables and ran methods.

Here, we will be creating applets exclusively and therefore we will need to know.

1. When to use applets
2. How an applets works.
3. What sort of features an applets has, and
4. where to start when we first create our own applets.

First of all, let us consider the situations when we might need to use applets.

1. When we need something dynamic to be included in the display of a web page.

For example, an applet that displays daily sensitivity index would be useful on a page that lists share prices of various companies or an applet that displays a bar chart would add value to a web page that contains data tables.

2. When we require some “flash” outputs.

For example ,applets that produce sounds, animations or some special effects would be useful when displaying certain pages.

3. When we want to create a program and make it available on the Internet for us by others on their computers.

Before we try to write applets, we must make sure that Java is installed properly and also ensure that either the Java applet viewer or a Java-enabled browser is a available. The steps involved in developing and testing in applet are:

1. Building an applet code(.java file)
2. Creating an executable applet(.class file)
3. Designing a web page using HTML tags
4. Preparing <APPLET>tag
5. Incorporating <APPLET>tag into the web page
6. Creating HTML file
7. Testing the applet code

4.2.4 BUILDING APPLET CODE :

It is essential that our applet code uses the services of two classes, namely, Applet and Graphics from the Java class library .The Applet class which is contained in the java.applet package provides life and behavior to the applet through its methods such as init(),start(),and point().Unlike the applications, where Java calls the main() method directly to initiate the execution of the program, when an applet is loaded, Java automatically calls a series of Applet class methods for staring, running, and stopping the applet code. The Applet class therefore maintains the lifecycle of an applet.

The paint() method of the Applet class, when it is called, actually displays the result of the applet code on the screen. The output may be text, graphics, or sound. The paint()method, which requires a Graphics object as an argument ,it is defined as follows:

Public void paint(Graphics g)

This requires that the applet code imports the java.awt package that contains the Graphics class. All output operations of an applet are performed using the methods defined in the Graphics class. It is thus clear from the above discussion that an applet code will have a general format as shown below:

```
import java.awt.*;
import java.applet.*;
……
……
public class appletclassname extends Applet
{
…….
……..
public void paint(Graphics g)
{
……
…… // Applet operation code
……
}
……
……
}
```

The appletclasssname is the main class for the applet. When the applet is loaded ,Java creates an instance of this class,and then a series of Applet class methods are called on that instance to execute the code.

Program: The HelloJava applet

```
import java.awt.*;
import java.applet.*;
public class HelloJava extends Applet {
public void paint(Graphics g)
{
g.drawString("Hello Java",,);
```

}

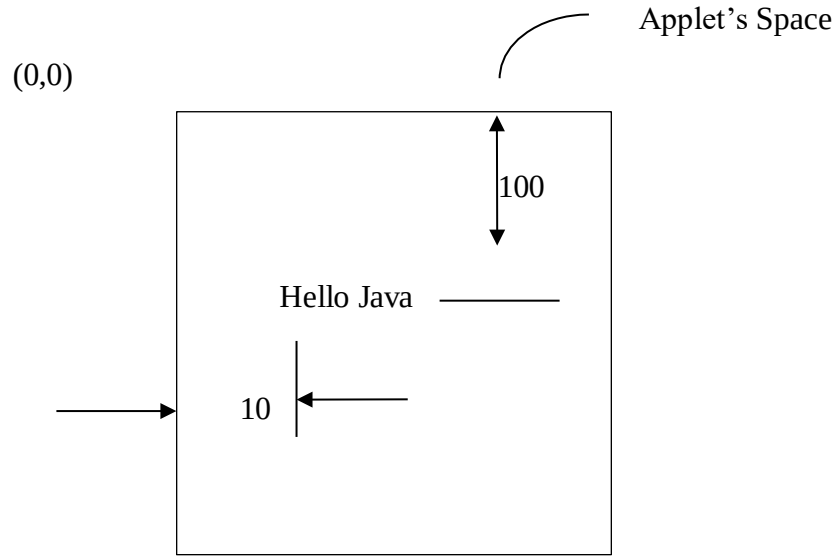
The applet contains only one executable statement.

g.drawString("Hello Java",10,100);

which ,when executed,draws the string

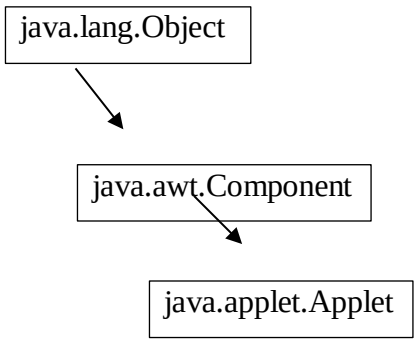
Hello Java

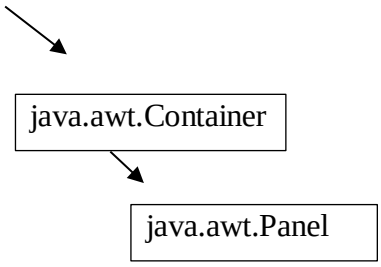
at the position 10,100 (pixels)of the applet's reserved space as shown figure



Remember that the applet code in above program, should be saved with the file name HelloJava.java in a java subdirectory. Note the public keyword for the class HellloJava.Java requires that the main applet class be declared public.

Remember that Applet class itself is a subclass of the Panel class ,which is again a subclass of the Container class and so on as shown in figure. This shows that the main applet class inherits properties from a long chain of classes. An applet can, therefore, use variables and methods from all these classes.

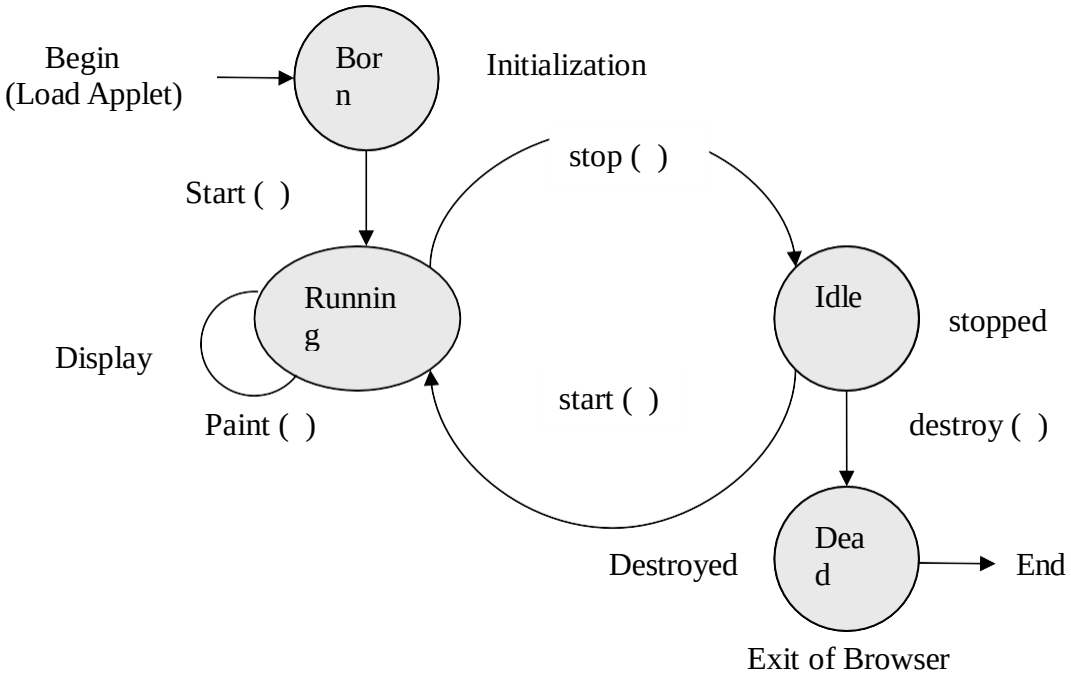




4.2.5 APPLET LIFE CYCLE :

Every Java applet inherits a set of default behaviors from the Applet class. As a result, when an applet is loaded ,it undergoes a series of changes in its states in figure. The applet states include:

Applet's stable transition diagram



1. Born on initialization state
2. Running state
3. Idle state
4. Dead or destroyed state

1. Initialization State :

Applet enters the initialization state when it is first loaded .This is achieved by calling the init( ) method of Applet class. The applet is born. At this stage, we may do the following ,if required.

- Create objects needed by the applet
- Set up initial values
- Load images or fonts
- Set up colors

The initialization occurs only once in the applet's life cycle. To provide any of the behaviors mentioned above, we must override the init( ) method:

```
public void init()
{
……..
….....(Action)
}
```

2. Running State :

Applets enters the running state when the system calls the start() method of Applet class. This occurs automatically after the applet is initialized. Starting can also occur if the applet is already in 'stopped'(idle)state. For example, we may leave the Web page containing the applet temporarily to another page and return back to the page. This again starts the applet running. Note that, unlike init( ) method, the start()method may be called more than once. We may override the start( ) method to create a thread to control the applet.

```
public void start()
{
        ……..
        … .....(Action)
        ……..
}
```

3. Idle or Stopped State :

An applet becomes idle when it is stopped from running. Stopping occurs automatically when we leave the page containing the currently running applet. We can also do so by calling the

stop( ) method explicitly. If we use a thread to run the applet, then we must use stop( ) method to terminate the thread.

We can achieve this by override the stop() method:

```
public void stop( )
{
        ……  (Action).
        …….
}
```

4. Dead State :

An applet is said to be dead when it is removed from memory. This occurs automatically by invoking the destroy() method when we quit the browser. Like initialization, destroying stage occurs only once in the applet's life cycle. If the applet has created any resources, like threads, we may override the destroy() method to clean up these resources.

```
public void destroy
{
        …….
        ……  (Action).
        …….
}
```

5. Display State :

Applet moves to the display state whenever it has to perform some output operations on the screen. This happens immediately after the applet enters into the running state. The paint( ) method is called to accomplish this task. Almost every applet will have a paint( ) method. Like other methods in the life cycle, the default version of paint() method does absolutely nothing. We must therefore override this method if we want anything to be displayed on the screen.

```
public void paint(Graphics g)
{
        ……
```

…… (Display statements)

……

}

It is to be noted that the display state is not considered as a part of the applet's life cycle. In fact ,the paint() method is defined in the applet class, It is inherited from the Component class, a super class of a applet.

4.2.6 CREATING AN EXECUTABLE APPLET :

Executable applet is nothing but the .class file of the applet, which is obtained by compiling the source code of the applet. Compiling an applet is exactly the same as compiling an application. Therefore ,we can use the Java compiler to compile the applet.

Here are the steps required for compiling the HelloJava applet.

1. Move to the directory containing the source code and type the following command: javac HelloJava.java
2. The compiled output file called HelloJava.class is placed in the same directory as the source.
3. If any error message is received, then we must check for errors, correct them and compile the applet again.

4.2.7 DESIGNING A WEB PAGE :

Recall the Java applets are programs that reside on web pages. In order to run a Java applet, it is first necessary to have a web page that references that applet.

A web page is basically made up of text and HTML tags that can be interpreted by a Web browser or an applet viewer. Like Java source code, it can be prepared using any ASCII text editor. A web page is also known as HTML page or HTML document. Web pages are stored using a file extension .html such as MyApplet.html. Such files are referred to as HTML files.HTML files should be stored in the same directory as the compiled code of the applets.

A pointed out earlier, Web pages include both text that we want to display and HTML tags(commands)to Web browsers. A Web page is marked by an opening HTML tag <HTML> and a closing HTML tag</HTML>and is divided into the following three major sections:

1. Comment section (Optional)

2. Head section (optional)
3. Body section

A web page outline containing these three sections and the opening and closing HTML tags is illustrated in the figure.

1. Comment Section :

This section contains comments about the web page. It is important to include comments that tell us what is going on in the Web page. A comment line begins with a <! And ends with a >.Web browsers will ignore the text enclosed between them. Although comments are important, they should be keep to a minimum as they will be downloaded along with the applet. Note that comments are optional and can be included anywhere in the web page.

2. Head section :

The head section is defined with a starting <HEAD>tag and a closing </HEAD> tag. This section usually contains a title for the web page as shown below:

```
<HTML>
<HEAD>
        <TITLE> WELCOME TO JAVA APPLETS </TITLE>
</HEAD>
```

The text enclosed in the tags <TITLE> and </TITLE> will apper in the title bar of the web browser when it displays the page. The head section is also optional.

Comment
Section

Head
Section
Body
Section

Note that tags <…….> containing HTML commands usually appear impairs such as <HEAD> and </HEAD>,and <TITLE> and </TITLE>.A slash(/)in a tag signifies the end of that tag section.

3. Body Section :

After the head section comes the body section. We call this as body section because this section contains the entire information about the Web page and its behavior. We can set up many options to indicate how our page must appear on the screen(like colour, location, sound, etc.,).Shown below is a simple body section:

```
<BODY>
    <CENTER>
        <H1> WELCOME TO THE WORLD OF APPLETS </H1>
    </CENTER>
    <BR>
    <APPLET …>
    </APPLET>
</BODY>
```

The body shown above contains instructions to display the message

WELCOME TO THE WORLD APPLETS

Followed by the applet output on the screen. Note that the <CENTER> tag makes sure that the text is centered and <H1> tag causes the text to be of the largest size. We may use other heading tags <H2> to <H6> to reduce the size of letters in the text.

4.2.8 APPLET TAG :

Note that we have included a pair of <APPLET…>and </APPLET> tags in the body section discussed above. The <APPLET..>tag supplies the name of the applet to be loaded and tells the browser how much space the applet requires. The ellipsis in the tag <APPLET… >indicates that it contains certain attributes that must specified. The <APPLET> tag given below specifies the minimum requirements to place the HelloJava applet on a Web page:

```
<APPLET
        CODE = helloJava.class
        WIDTH = 400
        HEIGHT = 200  >
</APPLET>
```

This HTML code tells the browser to load the complied Java applet HelloJava.class, which is in the same directory as the HTML file. And also specifies the Display area for the applet output as 400 pixels width and 200 pixels height. We can make this display area appear in the centre of the screen by using the CENTER tags shown as follows:

```
<CENTER>
        <APPLET>
         …….
         </APPLET>
</CENTER>
```

Note that <APPLET> tag discussed above specifies three things:

1. Name of the applet
2. Width of the applet (in pixels)
3. Height of the applet (in pixels)

4.2.9 ADDING APPLET TO HTML FILE :

Until now, we can put together the various components of the web page and create a file known as HTML file. Insert the <APPLET> tag in the page at the place where the output of the applet must appear. Following is the content of the HTML File that is embedded with the

<APPLET> tag of our HelloJava applet.

```
<HTML>
<! This page includes a welcome title in the title bar and also displays a welcome message.Then it specifies the applet to be loaded and executed.
>
<HEAD>
    <TITLE>
        Welcome to Java Applets
    </TITLE>
</HEAD>
<BODY>
    <CENTER>
        <H1> Welcome to the World of Applets </H1>
    </CENTER>
    <BR>
    <CENTER>
        <APPLET
            CODE = HelloJava.class
            WIDTH = 400

            HEIGHT = 200 >
        </APPLET>
    </CENTER>
</BODY>
</HTML>
```

Note: We must name this file as HelloJava.html and save it in the same directory as the complied applet.

4.2.10 RUNNING THE APPLET :

Now that we have created applet files as well as the HTML file containing the applet, we must have the following files in our current directory:

Hellojava.java

Hellojava.class

Hellojava.html

To run an applet,we require one of the following tools;

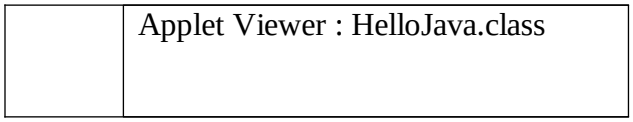
1. Java-enabled Web browser(such as HotJava or Netscape)
2. Java appletviewer

If we use a Java-enabled Web browser, we will be able to see the entire Web page containing the applet. If we use the applet viewer tool ,we will only see the applet output. Remember that the applet viewer is not a full-fledged web browser and therefore it ignores all of the HTML tags except the part pertaining to the running of the applet.

The appletviewer is available as a part of the Java Development Kit that we have been using so far. We can use it to run our applet as follows:

Appletviewer HelloJava.html

Notice that the argument of the appletviewer is not the .java file or the .class file ,but rather .html file . The output of our applet will be as shown below:



4.2.11 MORE ABOUT APPLET TAG :

We have used the <APPLET > tag in its simplest form. In its simplest form, it merely creates a space of the required size and then displays the applet output in that space. The syntax of the <APPLET>tag is a little more complex and includes several attributes that can help us better integrate our applet into the overall design of the web page. The syntax of the <APPLET> tag in full form is shown as follows:

<APPLET

[CODEBASE =codebase_URL]

CODE = AppletFileName.class

[ALT = alternate_text]

[NAME = applet_instance_name]

WIDTH = pixels

HEIGHT = pixels

[ALIGN =alignment]

[VSPACE =pixels]

[HPACE = pixels]

>

[< PARAM NAME = name1 VALUE = value1>]

[< PARAM NAME = name2 VALUE = value2>]

…….

……..

[Text to be displayed in the absence of java]

</APPLET>

The various attributes shown inside [] indicate the options that can be used when intergrating an applet into a Web page.Note that the minimum required attributes are:

CODE = AppletFileName.class

WIDTH = pixels

HEIGHT = pixels

ATTRIBUTES OF APPLET TAG :

ATTRIBUTE	MEANING
CODE=AppletFileName.class	Specifies the name of the applet class to be loaded .That is ,the name of the already-compiled .class file in which the executable Java bytecode for the applet is stored. This attribute must be specified
CODEBASE =codebase_URL (Optinal)	Specifies the URL of the directory in which the applet resides. If the applet resides in the same directory as the HTML file ,then the CODEBASE attribute may be omitted entirely.
WIDTH=pixels HEIGHT =pixels	These attributes specify the width and height of the space on the HTML page that will be reserved for the applet.

NAME =applet_instance_name	A name for the applet may optionally be specified so that other applets on the page may refer to this applet. This facilitates inter-applet communication.
ALIGN =alignment (Optinal)	This optional attribute specifies where on the page the applet will appear. Possible values for alignment: TOP,BOTTOM,LEFT,RIGHT,MIDDLE,ABSMIDDLE, ABSBOTTOM,TEXTTOP,AND BASELINE.
HSPACE=pixels (Optional)	Used only when ALIGN is set to LEFT or RIGHT, this attribute specifies the amount of horizontal blank space the browser should leave surrounding the applet.
VSPACE=pixels (Optional)	Used only when some vertical alignment is specified with the ALIGN attribute (TOP,BOTTOM<etc.,) VSPACE specifies the amount of vertical blank space the browser should leave surrounding the applet.
ALT=alternate_text (Optional)	Non-Java browsers will display this text where the applet would normally go. This attribute is optional.

We summaries below the list of things to be done for adding an applet to a HTML document:

1. Insert an <APPLET>tag at an appropriate place in the web page.
2. Specify the name of the applet's .class file.
3. If the .class file is not in the current directory, use the codebase parameter to specify :
   The relative path if file is on the local system, or
   The Uniform Resource Locator(URL) of the directory containing the file if it is on a remote computer.
4. Specify the space required for display of the applet in terms of width and height in pixels.
5. Add any user-defined parameters using <PARAM>tags.
6. Add alternate HTML text to be displayed when a non-java browser is used.
7. Close the applet declaration with the </APPLET>tag.

4.2.12 PASSING PARAMETERS TO APPLETS:

We can supply user-defined parameters to an applet using <PARAM...>tags .Each <PARAM...>tag has a name attribute such as color, and a value attribute such as red. Inside the

applet code, the applet can refer to that parameter by name to find its value. For example, we can change the color of the text displayed to red by an applet by using a <PARAM..>tag as follows:

<APPLET …..>
<PARAM = color VALUE =”red”>
</APPLET>

Similarly ,we can change the text to be displayed by an applet by supplying new text to the applet through a <PARAM…>tag as shown below:

<PARAM NAME =text VALUE =”I love Java”>

Passing parameters to an applet code using <PARAM> tag is something similar to passing parameters to the main() method using command line arguments. To set up and handle parameters, we need to do two things:

1. Include appropriate<PARAM..>tags in the HTML document.
2. Provide Code in the applet to parse these parameters.

Parameters are passed on an applet when it is loaded. We can define the init()method in the applet to get hold of the parameters defined in the <PARAM>tags. This is done using the getParameter() method, which takes one string argument representing the name of the parameter and returns a string containing the value of that parameter.

Following Program shows another version of HelloJava applet.Compile it so that we have a class file ready.

Program:Applet HelloJava Program

```
import java.awt.*;
import java.applet.*;
public class HelloJavaParam extends Applet
{
String str;
public void init()
 {
str = getParameter("String");  // Receiving parameter value
if (str==null)
```

```
str = "Java";
str ="Hello"+ str;                    //Using the value
}
public void paint(Graphics g)

{
g.drawString(str,10,100);
}

}
```

Now ,let us create HTML file that contains this applet. Following Program shows a web page that passes a parameter whose NAME is “string ”and whose VALUE is “APPLET!” to the applet HelloJavaParam.

program: The HTML file for the HelloJavaParam applet

```
<HTML>
        <! Parameterized HTML file >
<HEAD>
<TITLE> Welcome to java applets </TITLE>
</HEAD>
<BODY>
        <APPLET CODE =HelloJavaParam.class
            WIDTH =400
             HEIGHT =200>
        <PARAM NAME =”string ”
            VALUE =”Applet!”>
</APPLET>
        </BODY>
</HTML>
```

Save this file as HelloJAvaParam.html and then run the applet using the applet viewer as follows:

```
appletviewer HelloJavaParam.html
```

4.2.13 ALIGNING THE DISPLAY :

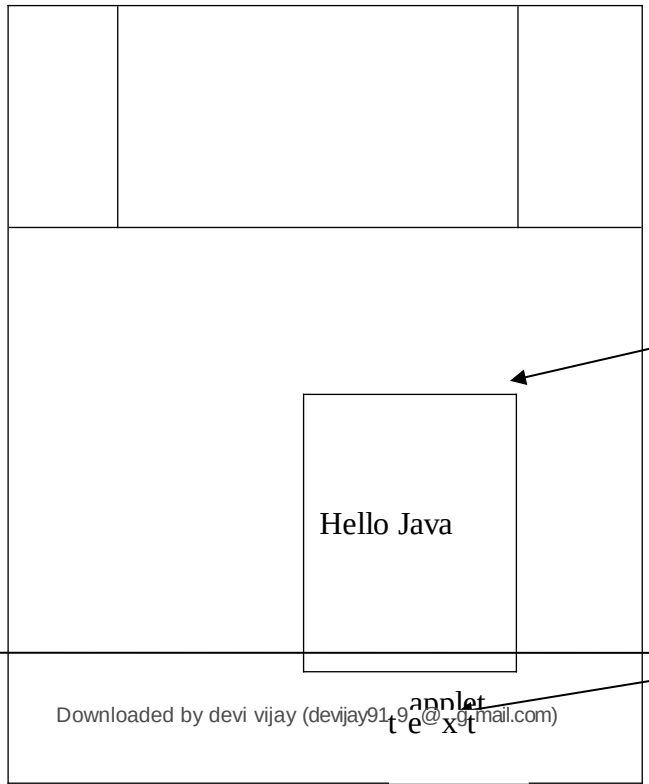
We can align the output of the applet using the ALIGN attribute.This attribute can have one of the nine values:

LEFT, RIGHT, TOP, TEXT TOP, MIDDLE, ABSMIDDLE, BASELINE, BOTTOM, ABSBOTTOM

For example ALIGN = LEFT will display the output at the left margin of the page.All text that follows the ALIGN in the Web page will be placed to the right of the display.Following Program shows a HTML file our HelloJava applet.

```
<HTML>
<HEAD>
<TITLE> Here is an applet </TITLE>
</HEAD>
<BODY>
        <APPLET CODE =HelloJava.class
            WIDTH =400
             HEIGHT =200
            ALIGN = RIGHT >
</APPLET>
        </BODY>
</HTML>
```

The alignment of applet will be seen and appreciated only when we run the applet using a Java capable browser. Figure shows how an applet and text surrounding it might appear in a capable browser. All Java the text following the applet appears to the left of that applet.



4.2.14 MORE ABOUT HTML TAGS :

We have seen and used a few HTML tags.HTML supports a large number of tags that can be used to control the style and format of the display of web pages.

Important HTML tags and functions

Tag	Function
<HTML>…..</HTML>	Signifies the beginning and end of a HTML file.
<HEAD>…..</HEAD>	This tag may include details about the web page. Usually contains <TITLE>tag within it.
<TITLE>….</TITLE>	The text contained in it will appear in the title bar of the browser.
<BODY>…..</BODY>	This tag contains the main text of the web page. It is the place where the <APPLET>tag is declared.
<H1>…….</H1>	Header tags .Used to display headings.<H1> creates the largest font header ,while<H6>creates the smallest one.
<H6>…..<H6>	
<CENTER>..... </CENTER>	Places the text contained in it at the center of the page.
<APPLET……>	<APPLET….> tag declares the applet details as its attributes.
<APPLET>….</APPLET>	May hold optionally user defined parameters using<PARAM>tags
<PARAM>	Supplies user-defined parameters. The <PARAM>tag needs to be placed between the <APPLET> and </APPLET> tags. We can use as many different <PARAM> tags as we wish for each applet.
<B>…..</B>	Text between these tags will be displayed in bold type.
 	Line break tag. This will skip a line. Does not have an end tag.

<P>	Para tag. this tag moves us to the next line and starts a paragraph of text. No end tag is necessary.
<IMG…>	This tag declares attributes of an image to be displayed.
<HR>	Draws a horizontal rule.
<A….></A>	Anchor tag used to add hyperlinks.
<FONT…>….</FONT>	We can change the color and size of the text that lies in between <FONT>and </FONT>tags using COLOR and SIZE attributes in the tag<FONT…>.
<!....>	Any text starting with a <!mark and ending with a >mark is ignored by the web browser. We may add comments here. A comment tag may be placed anywhere in a web page.

4.2.15 DISPLAYING NUMERICAL VALUES :

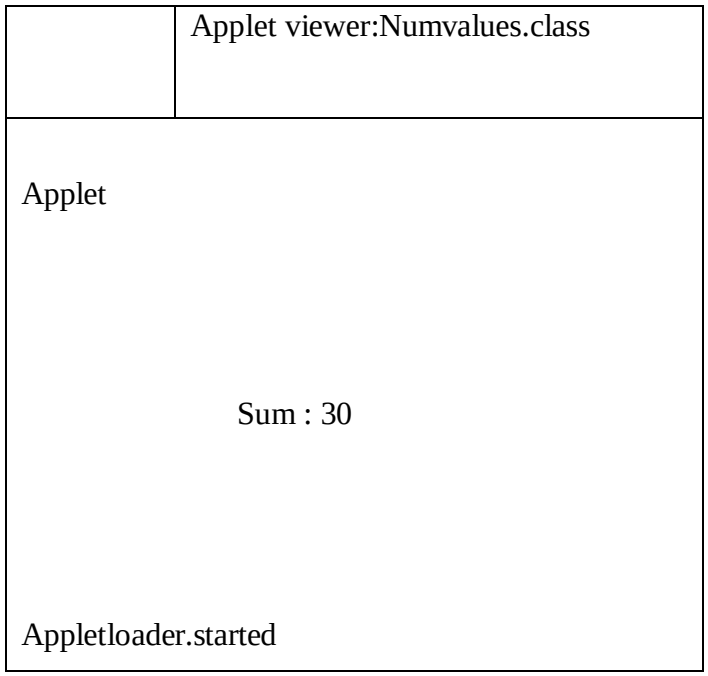
It applets ,we can display numerical values by first converting them into strings and then using the drawstring() method of Graphics class. We can do this easily by calling the ValueOf()method of String class. Following program illustrates how an applet handles numerical values.

```
import java.awt.*;
import java.applet.*;
public class NumValues extends Applet
{
public void paint(Graphics g)
{
int value1 =10;
int value2 =20;
int sum = value1+ value2;
String s = “Sum: ”+ String.valueOf(sum);
g.drawString(s,100,100);
}
```

}

The applet program when run using the following HTML file displays the output as shown in figure:

```
<html>
<applet
code=NumValues.class
width=300
height=300 >
</applet>
</html>
```



4.2.16 GETTING INPUT FROM THE USER :

Applets work in a graphical environment. Therefore ,applets treat inputs as text strings. We must first create an area of the screen in which user can type and edit input items(which may be any data type). We can do this by using the Textfield class of the applet package.

Once text fields are created for receiving input, we can type the values in the fields and edit them, if necessary.

Next step is to retrieve the items from the fields for display calculation, if any Remember, the text fields contain items in string form. They need to be converted to the right form, before

they are used in any computations.

The results are then converted back to strings for display. Following program demonstrates how these steps are implemented.

Program: Interactive input to an applet

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
import java.io.*;

public class UserIn extends Applet
{
TextField text1,text2;
public void init()
{
text1=new TextField(8);
text2=new TextField(8);
add(text1);
add(text2);
text1.setText("0");
text2.setText("0");
}
public void paint(Graphics g)
{
int x=0,y=0,z=0;
String s1,s2,s;
g.drawString("Input a number in each box",10,50);
try
{
s1=text1.getText();
x=Integer.parseInt(s1);
s2=text2.getText();
y=Integer.parseInt(s2);
}
```

```
catch(Exception e)
{
}
z=x+y;
s=String.valueOf(z);
g.drawString("The sum is",10,75);
g.drawString(s,100,75);
```

```
}
public boolean action(Event event,Object object)
{
repaint();
return true;
}
}
```

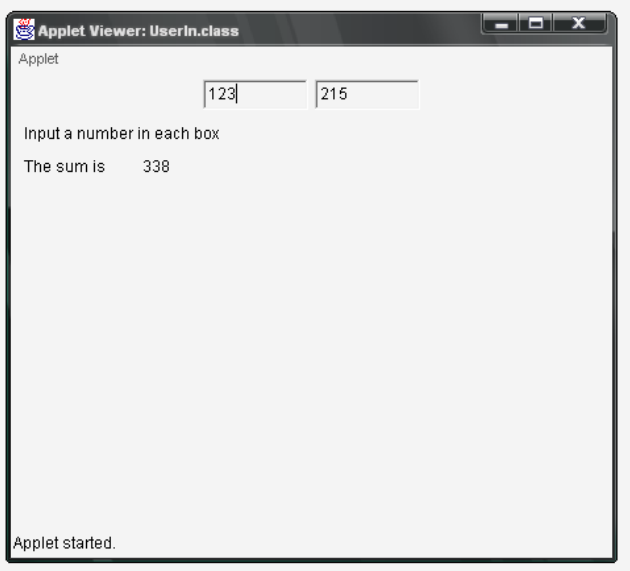
Run the applet UserIn using the following steps:

1. Type and save the program(.java file)
2. Compile the applet(.class file)
3. Write a HTML document(.html file)

```
<html>
    <applet
        code =UserIn.class
        width = 300
        height =200 >
    </applet>
</html>
```

4. Use the appletviewer to display the results

when the applet is up and running, enter a number into each text field box displayed in the applet area, and then press the Return Key .Now, the applet computes the sum of these numbers and displays the result as shown below:



Program analysis

The applet declares two TextField objects at the beginning.

TextField text1,text2;

These two objects represent text boxes where we type the numbers to be added.

Next ,we override the init() method to do the following:

1. To create two text field objects to hold strings(of eight character length).
2. To add the objects to the applet's display area.
3. To initialize the contents of the objects to Zero.

Then comes the paint(0 method where all the actions take place. First ,three integer variables are declared ,followed by three string variables. Remember, the numbers entered in the text boxes are in string from and therefore they are retrieved as strings using the getText() method and then they are converted to numerical values using parseInt(0 method of the Integer class.

After retrieving and converting both the strings to integer numbers, the paint (0 method sums them up and stores the result in the variable i.e. must convert the numerical value in z to a string before we attempt to display the answer. This is done using the ValueOf method of the String class.

4.3 GRAPHICS PROGRAMMING

One of the most important features of Java is its ability to draw graphics. we can write Java applets then draw lines, figures of different shapes, images and text in different fonts and styles. Every applet has its own area of the screen known as ***Canvas.***

4.3.1 THE GRAPHICS CLASS :

To draw a shape on the screen, we may call of the methods available in the graphics class. All the drawing methods have arguments representing end points, corners, or starting locations of a shape as values in the applet's coordinated system.

Methods	**Description**
clearRect ( )	Erases a rectangular area of the Canvas.
copyArea ( )	Copies a rectangular area of the canvas to another area.
drawArc ( )	Draws a hollow arc.
drawLine ( )	Draws a straight line.
drawOval ( )	Draws a hollow oval.
drawPolygon ( )	Draws a hollow polygon.
drawRect ( )	Draws a hollow rectangle.
drawRoundRect ( )	Draws a hollow rectangle with rounded corners.
drawstring ( )	Display a text string.
fillArc ( )	Draws a filled arc.
fillOval ( )	Draws a filled oval.
fillPolygon ( )	Draws a filled polygon.
fillRect ( )	Draws a filled rectangular.
fillRoundRect ( )	Draws a filled rectangle with rounded corners.
getColor ( )	Retrieves the current drawing color.
getFont ( )	Retrieves the currently used font.
getFontMetrics ( )	Retrieves the information about the current font.

setColor ( )	Sets the drawing colour.
setFont ( )	Sets the font.

4.3.3 Lines and Rectangles :

The **drawLine ( )** method takes two pair of co-ordinates (x1,y1) and (x2,y2) as arguments and draw a line between them.

For eg: drawing a straight line :

g.drawLine (10,10, 50,50) ;

The **g** is the Graphics object passed to paint ( ) method.

Similarly ,

1. we can draw a rectangle using the **drawRect( )** Method.
2. We can draw solid box by using the method **fillRect( )**
3. We can draw rounded rectangle using the method **drawRoundRect( ) & fillRoundRect( ).**

```
public class LinesRect extends Applet
{
   public void paint(Graphics g ){
        g.drawLine( 10 , 10 , 50 , 50 ) ;
        g.drawRect( 10 , 60  , 40 , 30 ) ;
        g.fillRect( 60  , 10,   30 , 80 ) ;
        g.drawRectRound( 10 , 100  , 80  , 50 , 10 , 10) ;
        g.fillRectRound( 20 , 110  , 60  , 30 , 5 , 5) ;
   }
}
```

drawArc()

Arcs can be drawn using the following two methods

void drawArc(int top, int left, int width, int height, int startAngle, int sweepAngle);

void fillArc(int top, int left, int width, int height, int startAngle, int sweepAngle);

the arc is drawn from startAngle and the angular distance is specified by sweepAngle. If sweepAngle is positive the arc is drawn counterclockwise and if the angle is negative the arc is drawn clockwise.

```
import java.awt. *;
import java.applet.*;

/*<applet code  = "Arcs " width = 300 height = 200 >
 </applet >
 */
public class Arcs extends Applet {
    public void paint(Graphics g ){
       g.drawArc(10, 40, 70, 70, 0, 75);
       g.fillArc(10, 40, 70, 70, 0, 75);
       g.drawArc(10, 40, 80, 70, 0, 75);
   }
}
```

SIZING GRAPHICS

Sometimes we wish to size a graphics object to the size of the window. To do this, we can first get the current dimensions of the window by getSize() . **this** returns the dimensions of window encapsulated in Dimensions object.

We will see an example of an applet that starts as a 200 * 200 pixel and grow by 25 pixel with each mouse click until the applet gets larger by 500 * 500. When it reaches that size the next mouse click will make the applet to 200 * 200 and the process starts again.

```
import java.applet.* ;
import java.awt.* ;
import java.awt.event.* ;
/* <applet code =  "Resize " width = 300 height = 200 >
  </applet>
*/
public class Resize extends Applet {
        final  int inc = 25 ;
        int max  = 500 ;
        int min = 200 ;
        Dimension d ;
    public  Resize () {
     addMouseListener(new MouseAdapter() {
        public void mouseReleased(MouseEvent me ){
            int w = (d.width + inc ) > max  ? min : (d.width + inc ) ;
           int h =  (d.height + inc ) > max  ? min : (d.height + inc ) ;
           setSize ( new Dimension ( w, h ) ;
        }
     } ) ;
    }
     public void paint(Graphics g ){
        d = getSize() ;
        g.drawLine( 0, 0, d.width – 1, d.height – 1);
        g.drawLine(0, d.height –1, d.width – 1, 0 );
     }
}
```

FONT MANIPULATION

AWT supports multiple types of fonts. Fonts are encapsulated by **Font** class **.**

Methods in Font class :

Method	Description
static Font decode(String str)	Returns a font given its name
boolean equals(Object FontObj)	Returns true if the invoking object contains the same font as specified FontObj
String getFamily()	Returns the name of the Font family associated with the invoking font
String getFontName()	Returns the face name of the invoking font
String getName()	Returns the logical name of the invoking font
int getSize()	Returns the size of the invoking font
int getStyle()	Returns the style value of the font
boolean isBold()	Returns true if the font includes bold style
boolean isItalic()	Returns true if the font includes italic style
Boolean isPlain()	Returns true if the font includes plain style

Ex –

```
import java.awt.* ;
import java.applet.* ;
public class FontInfo extends Applet{
  public void paint(Graphics g){
    Font f = g.getFont() ;
```

```
   String family = f.getFamily() ;
   String name = f.getName() ;
   String style = f.getStyle() ;
   int size = f.getSize) ;
   String msg = "family " + family + " " +" Font type" + name + "Style " + style  +
                    "size" + size ;
   g.drawstring(mgs, 10, 20) ;
  }
}
```

IMAGEOBSERVER

ImageObserver is an interface to receive notification as an image is generated.
It informs the user about the current status of applet which is downloaded.
It contains only one method : imageUpdate()

boolean imageUpdate(Image imgObj, int flags, int left, int top, int width, int height)

imgObj is the image being loaded, flags communicate the status of the update report. The integers top, left. Width and height specifies the rectangle that contains values depending on the value of the flags. imageUpdate() returns true if there is more image to process and returns false if loading of the image is completed. The flags parameter may contain one or more bit flags. The bit flags are described below

Flag	***Meaning***
WIDTH	Contains the width of the image
HEIGHT	Contains the height of the image
SOMEBITS	More pixels are needed to draw the image
ALLBITS	Image is now complete. The top, left, height and

	Width parameters are not used
ERROR	An error has occurred to and image that was being tracked
ABORT	The image that was tracked was aborted before it was complete

Usage of imageUpdate() method:

```
public Boolean imageUpdate(Image img, int flags, int x, int y, int w, int h) {
  if ( ( flags & ALLBITS ) == 0) {
            System.out.println("still processing image");
             return true ;
       }else{
            System.out.println("Processing is complete");
            return false ;
       }
}
```

IMAGE LOADER

Image can be loaded using the getImage() method defined by the Applet class.

```
Image getImage(URL url) ;
Image getImage(URL url, String imageName) :
```

These methods were described in the Applet class Once you have an image it can be displayed using drawImage() method which belongs to Graphics class.

```
boolean drawImage(Image imgObj, int left, int top, ImageObserver imgObj);
```

imgObj is the reference to the class the implements ImageObserver interface.

Suggested Question

1. **Explain the life cycle of Applet.**
2. **Discuss briefly about the Graphics programming.**
3. **How to manage errors and exceptions**
4. **How to perform applet programming.**
5. **Explain font class with suitable examples**

UNIT-V

Managing Output /Input files in Java: concepts of Streams-Stream Classes – Byte Stream Classes – character Stream Classs – Using Streams – I/O classes – File Class – I/O exceptions – Creation of Files- Reading / Writing characters , Byte handling primitive data types – Random access files

---

Managing output & output Files in Java:

We have used variables and arrays for storing data inside the programs. These approach posses the following problems.

1. The data is lost either when a variable goes out of scope or when the program is terminated, that is, the storage id temporary .
2. Its difficult to handle large volumes of data using variables & arrays.

We can over this problems by storing data on secondary storage devices such as floppy disk or hard disk. The data is stored in these devices using the concept of files.ata stored in files is often called persistant data.

File

File class doesn't specify how information is retrieved, it describes the properties of a file. A File object is used to obtain or manipulate information associated with a disk file such as permissions, date, time etc

Constructors to create File objects

File(String directoryPath) ;
File(String directoryPath , String filename) ;
File(File dirObj, String filename) ;

DirectoryPath specifies path of the file , dirObj is the File object that specifies the

Ex –

```
import java.io.File ;
class FileDemo {
static void p (String s){
System.out.println(s);
    }
public static void main(String args [ ] ){
 File f1 = new File("/java/Copyright");
 p("Filename :"+ f1.getName());
 p("Path :" + f1.getPath() );
 p("parent :" + f1.getParent() );
 p(f1.exists() ? "exists " : "file doesn't exists");
p(f1.canWrite() ? "is writeable" : "is not writeable");
p(f1.isDirectory() ? " " : "not a directory" );
 p(f1.isFile() ? "is a normal file " : "may be a named pipe");
 p("file size " + f1.length() );
   }
}
```

isFile() returns true if it is a file and false if it is a directory.

Method	*Description*
void deleteOnExit()	Removes the file associated with the invoking object when JVM terminates
boolean isHidden()	Returns true if the invoking file is hidden
setReadOnly()	Sets the invoking file to read-only
boolean renameTo (File newname)	Renames the file to the name specified by newname

boolean delete() deletes the file. Deletes directory if it is empty

Directories

A directory is a File that contains a list of other files and directories. When you create a File object and if it is a directory the isDirectory() method will return true. You can call list() method to extract the list of the files and directories inside.

```
public class Dirlist {
 public static void main(String args [ ]){
   String dirnam = "/java" ;
            File f1 = new File(dirname);
           if(f1.isDirectory()){
    System.out.println("Directory " +dirname);
   String s [ ] = f1.list [ ] ;
  for(int i = 0 ; i < s.length ; i ++ ){
     File f = new File(dirname + "/" + s [i ] ) ;
   if ( f.isDirectory()) {
   System.out.println( s[ i ]  + " is a directory ") ;
   }else {
  System.out.println (s[ i ] + " is a file " ) ;
  }
}
            else {
System.out.println(dirname + " is not a directory");
            }
          }
        }
```

FilenameFilter

To limit the number of files returned by the list() method to include only those files that match a certain filename pattern, we use the other form of list() as

String [ ] list(FilenameFilter FFobj) ;

FFobj implements FilenameFilter interface . this interface has only one method accept ()

boolean accept(File directory , String filename) ;

This returns true for files in the directory specified by directory that should be Included in the list and true if the files are to be excluded .

Ex –

```
import java.io.* ;
public class OnlyExt implements FilenameFilter {
  String ext ;
   public OnlyExt(String ext) {
     this.ext = "."  + ext  ;
   }
  public boolean accept(File dir, String name ){
       return name.endswith(ext );
 }
}

import java.io.* ;
class DirListOnly {
      public static void main(String args [ ] ){
          String dirname = "/java" ;
```

```
        File f1 = new File(dirname) ;
        FilenameFilter only = new OnlyExt("html") ;
        String s[ ] = f1.list(only) ;
        for(int i = 0 ; i < s.length ; i ++ ){
          System.out.println(s[ i ]) ;
         }
    }
}
```

The above program is to list files with extension .html

FileInputStream

This class creates an InputStream to read bytes from a file. The constructors are

FileInputStream(String filepath) ;
FileInputStream(File fileobj) ;

Ex
FileInputStream f = new FileInputStream("/file1.txt ");

File f1 = new File ("file1.txt");
FileInputStream f2 = new FileInputStream(f1) ;

method available() is used to determine the number of bytes remaining.

```
import java.io.*;
class FileInputStreamDemo {
    public static void main(String args [ ]){
        FileInputStream f = new FileInputStream( "Sample.java");
        System.out.println("number of bytes available " + f.available() );
```

```
        int size = f.available() ;
        for(int i = 0 ; i < size ; i ++ ){
            System.out.print( (char) f.read());
        }
    }
}
```

method read() is to extract bytes from the file

FileOutputStream

This class creates an OutputStream to write bytes to a file. The constructors are

FileOutputStream(String filepath) ;
FileOutputStream(File fileobj) ;
FileOutputStream(String filepath , boolean append) ;

These methods can throw IOException or SecurityException.

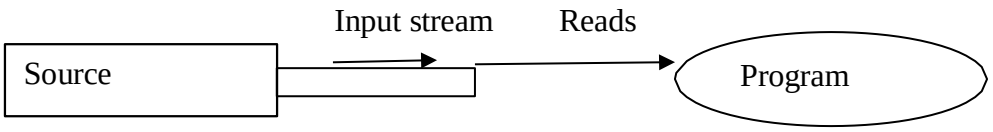
```
import java.io.* ;
class FileOutputDemo{
      public static void main(String args [ ] ) {
         String source = " creation of file output stream " ;
         byte b [ ]  =  source.getBytes() ;
        OutputStream f  = new FileOutputStream( "file1.txt ");
        for ( int i = 0 ; i < b.length ; i ++ ){
           f.write(b[ i ] );
         }
      }
}
```

the msg in the string source is written to the file file1.txt

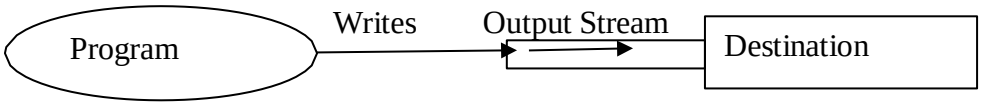
Concept of Stream

Java uses the concept of streams to represent the ordered sequence of data. A stream presents a uniform, easy to use, object oriented interface between the program and input and output devices

A stream in java is a path along which data flows. It has a sources and a destination. We can build a complex file processing sequence using a series of simple stream operations. This feature can be used to filter along the pipeline of streams so that we obtain data in a desired format.



Reading data into a program



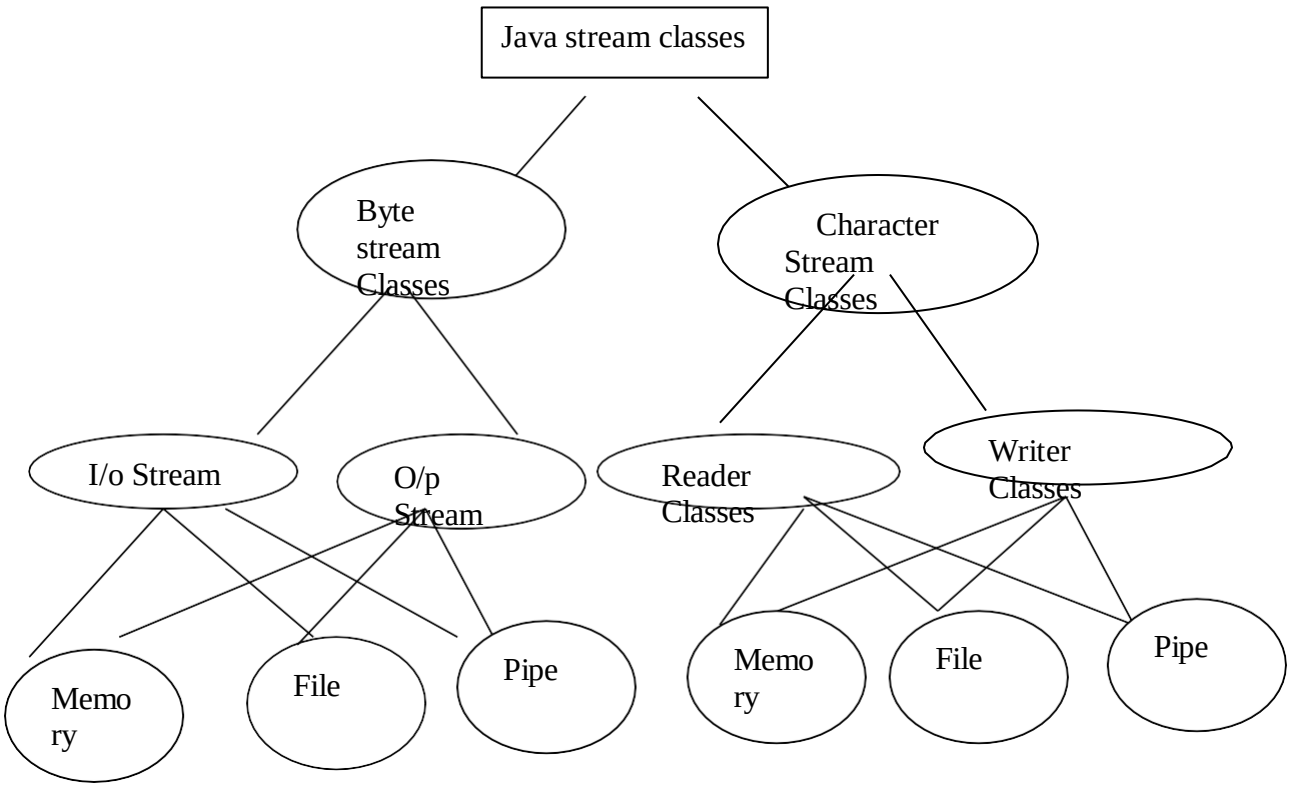
Writing data into a program

Ja va streams are classified into two basic types, namely input stream and output stream. An input stream extracts data from the source and sends to the program and output stream takes data from the program and send to the destination.,

Stream classes

The Java.io package contains a large number of stream classed that provide capabilities for processing all types of data . these classes may be categorized into two groups based on the data type on which they operate.

1. Byte stream classes that provide support for handling i/o operations on bytes.
2. Character stream classes that provide support for managing i/o operations on characters.



Byte Stream classes

Byte Stream Classes have been designed to provide functional features for creating and manipulating stream and files for reading and writing bytes. Java provides two types of byte stream classes. Input stream and output stream classes.

Input Stream classes

Input stream classes are used to read 8-bit byte include a super class known as input stream.

The Inputstream includes methods to perform the following tasks

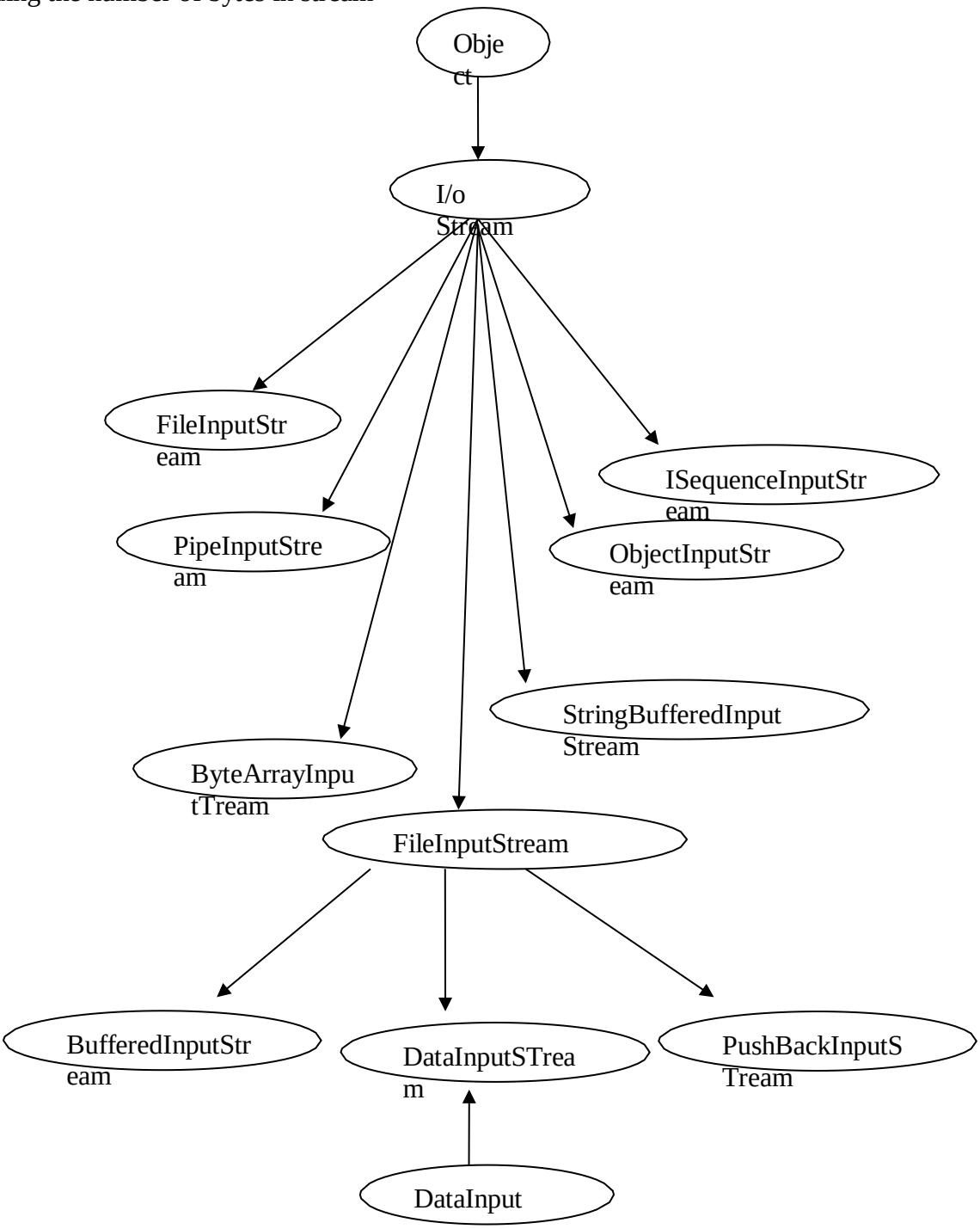
Reading bytes

Closing stream

Marking position in stream

Skipping ahead in stream

Finding the number of bytes in stream

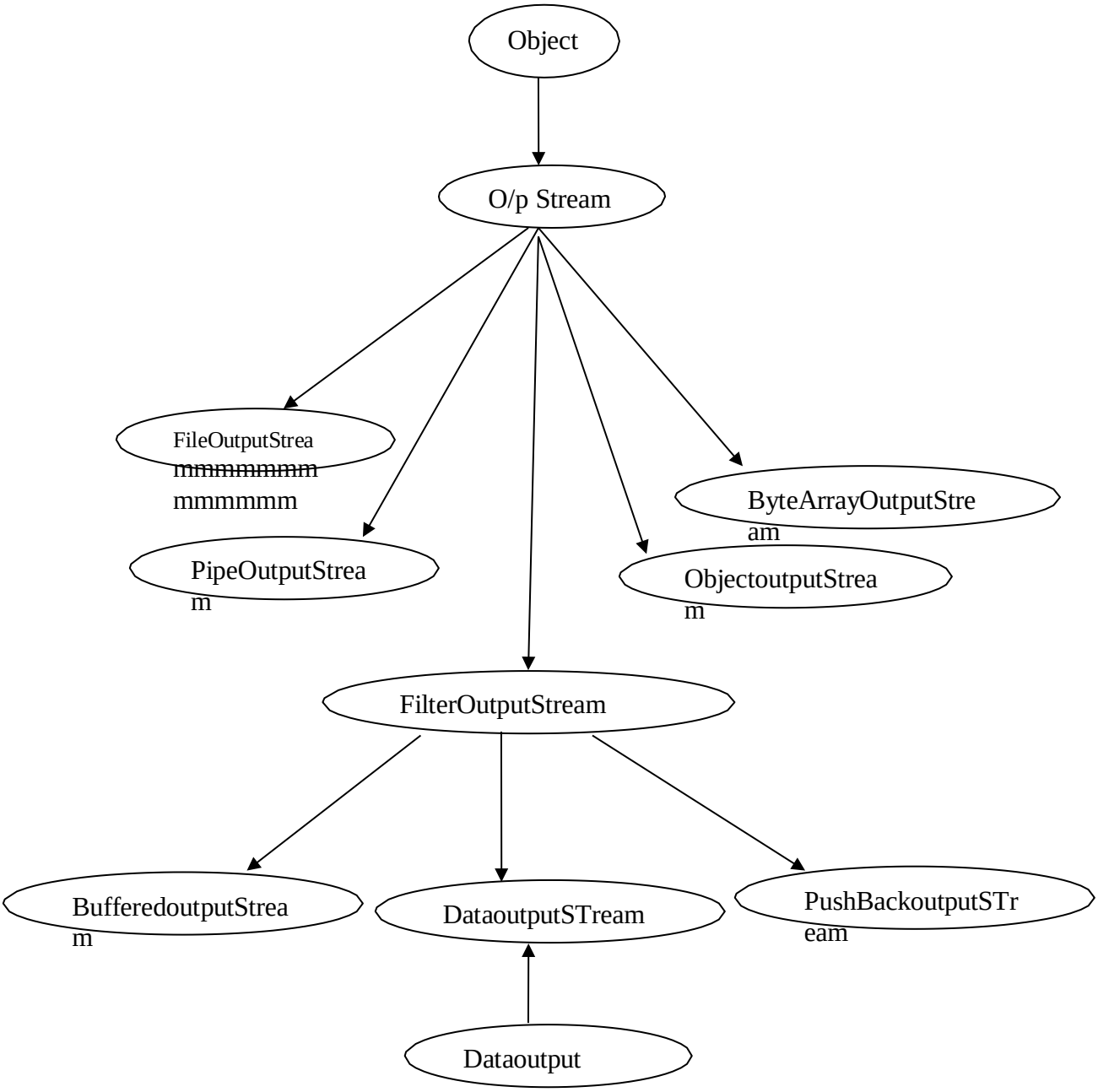


Output Stream Classes are derived from the base class output stream. Output stream is an abstract class and therefore we can not instantiate it. The outputstream includes methods to perform the following tasks

Writing bytes

Closing stream

Flushing stream

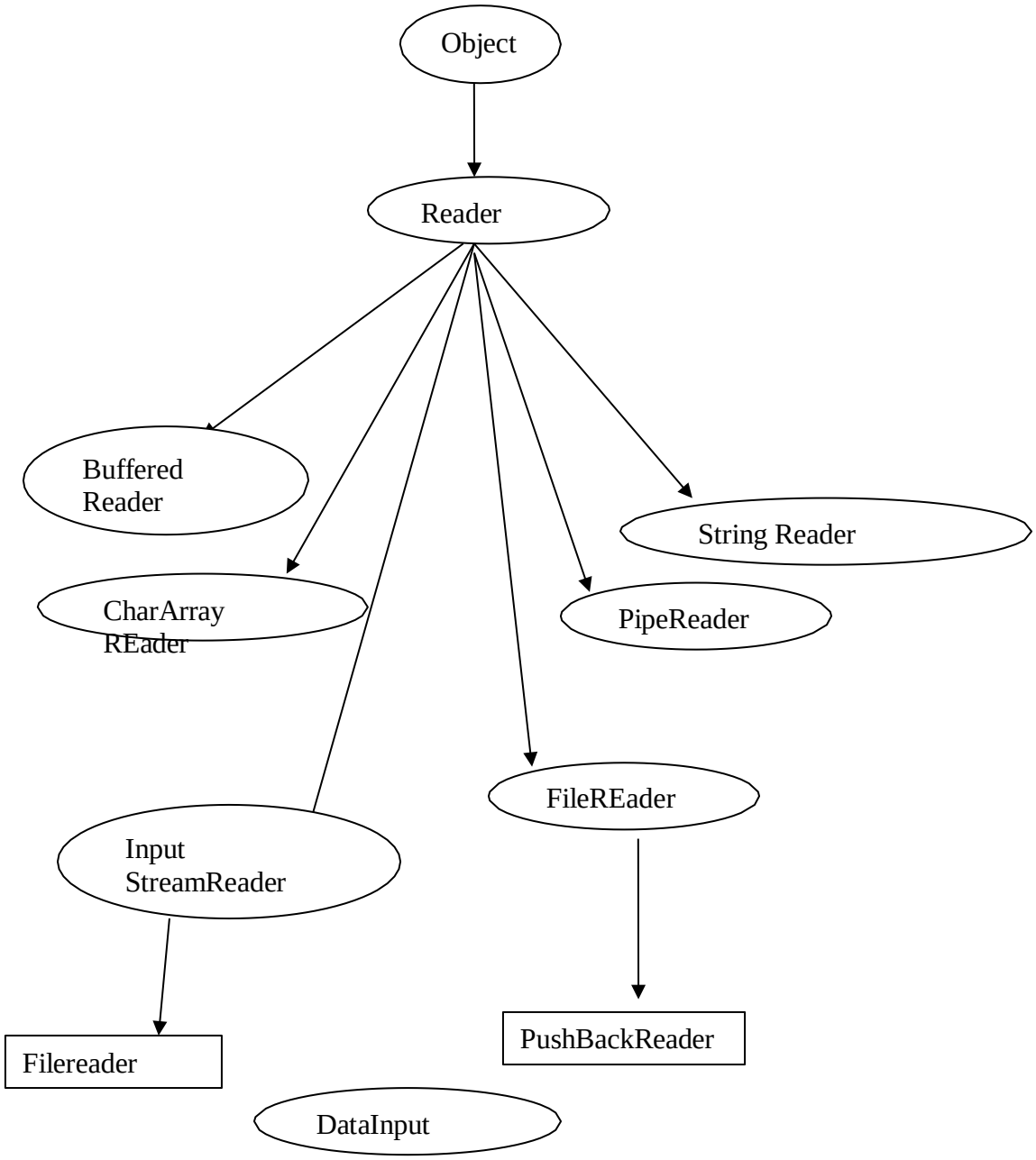


Character Stream Classes

Character Stream classes can be used to read an write 16-bit Unicode character. Ther e are two kind of Character Stream Classes are reader stream classes and writer stream classes.

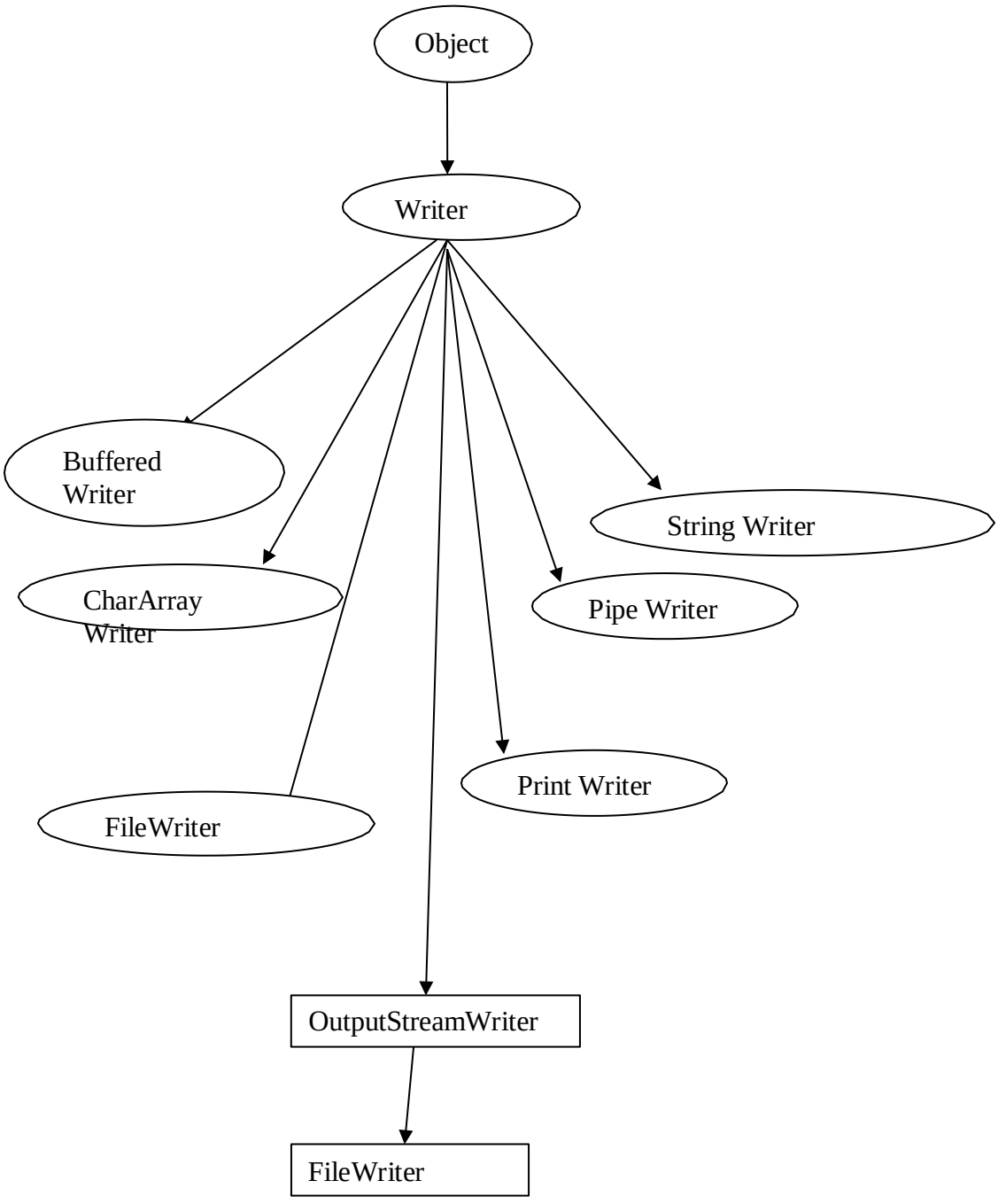
REAder Stream Classes

Reader stream classes are design ed to read character form the files. Reader stream use characters.



Writer Stream Classes

Writer stream classes are designed to write character. Writer stream use characters.



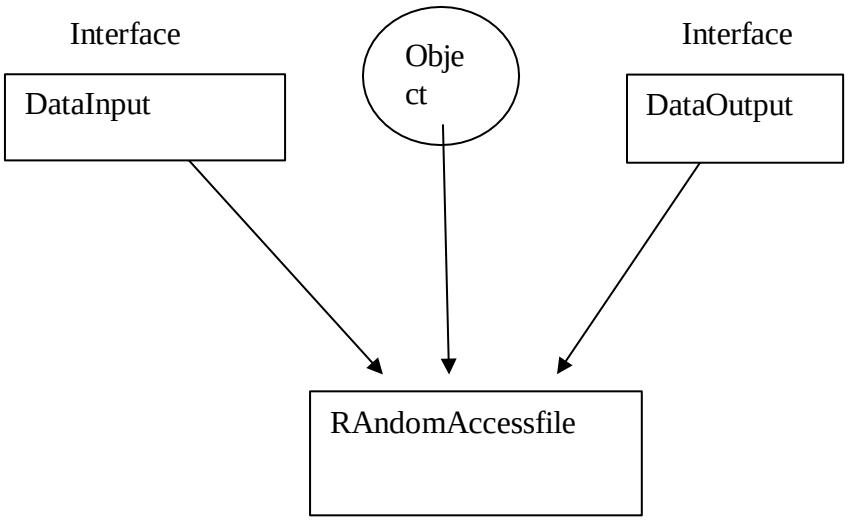
Using Streams

Task	Character Stream Class	Byte STreamclass
Buffering Input	BufferedREader	BufferedInputSTream
Translating BytesTream into a chacter stream	InputSTreamREader	none
Reading From files	FileREader	FileInputSTream
Reading From string	StringREader	StringBufferedInputSTream
Buffering output	BufferedWriter	BufferedoututSTream
Translating Character sTream into a byte stream	outputSTreamREader	none
writing to files	Filewriter	FileoutputSTream
writing to string	Stringwriter	none

Other Useful I/o Classes

Randomaccessfile enables us to read and write bytes, test and java data types to any location in a file

StreamTokenizer used for breaking up a stream of text from an input text file into tokens.



Using The File Class

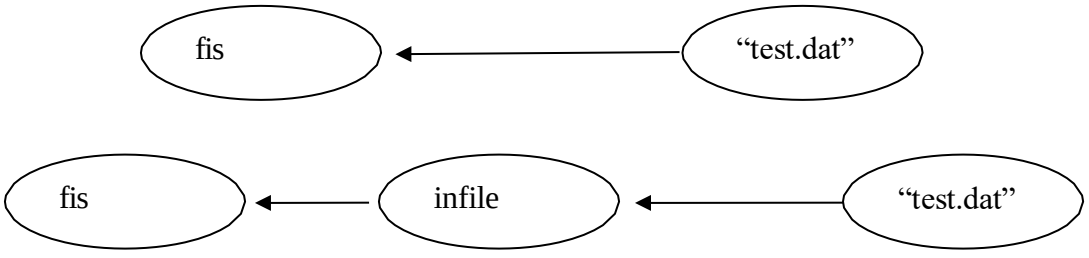
- Creating a file
- Opening a file
- Closing a file
- Deleting a file
- Getting name of a file
- Getting the size of a file
- Checking the existence of a file
- REanaming a file
- Checking whether the file is writable
- Checking whether the file is readable

I/O Exceptions

I/O Exception Class	Function
EOF function	Signal that an end of file or end of stream has been reached
FileNotFoundException	Inform that an file cannot be found
InterrruptedIOException	Warns tat an IO operation has been interrupted
IOException	Signals that an I/O exception of some sort has occurred.

Creation of Files

There are two ways to initializing the file stream objects. all of the constructors require that we provide the name of the file either directly or indirectly.



Reading / Writing characters

the two subclasses used for handling characters in files are File Reader and File Writer.

```
class copy
{
public static void main(String args[])
{
File infile  =new (“input.dat”);
File outfile  =new (“output.dat”);
ins = new FileReader(infile);
outs = new FileWriter(outfile);
int ch;
while(ch=ins.read())!=-1)
{
out.write(ch);
}
ins.close();
out.close();
}}
```

Reading / Writing Bytes

FileInputStream

This class creates an InputStream to read bytes from a file. The constructors are

FileInputStream(String filepath) ;
FileInputStream(File fileobj) ;

Ex
FileInputStream f = new FileInputStream(“/file1.txt “);

File f1 = new File (“file1.txt”);
FileInputStream f2 = new FileInputStream(f1) ;

method available() is used to determine the number of bytes remaining.

```
import java.io.*;
class FileInputStreamDemo {
     public static void main(String args [ ]){
        FileInputStream f = new FileInputStream( “Sample.java”);
       System.out.println(“number of bytes available “ + f.available() );
       int size = f.available() ;
       for(int i = 0 ; i < size ; i ++ ){
           System.out.print( (char) f.read());
       }
    }
}
```

method read() is to extract bytes from the file

FileOutputStream

This class creates an OutputStream to write bytes to a file. The constructors are

```
FileOutputStream(String filepath) ;
FileOutputStream(File fileobj) ;
FileOutputStream(String filepath , boolean append) ;
```

These methods can throw IOException or SecurityException.

```
import java.io.* ;
class FileOutputDemo{
      public static void main(String args [ ] ) {
        String source = “ creation of file output stream “ ;
        byte b [ ]  =  source.getBytes() ;
       OutputStream f  = new FileOutputStream( “file1.txt “);
       for ( int i = 0 ; i < b.length ; i ++ ){
         f.write(b[ i ] );
```

```
        }
      }
}
```

the msg in the string source is written to the file file1.txt

Handling Primitive data types

If we want to read/write the primitive data types such as integer and double , we can use filter classes as wrapper on exisiting input and output stream.

```
class readp
{
public static void main(String args[])
{
File p = new File("prim.dat");
FileOutputStream f0s = new FileOutputStream(p);
DataOutputStream d0s = new DataOutputStream(fos);
dos.writeInt(1999);
dos.writeDouble(234.4);
dos.close();
fos.close();
FileInputStream fis = new FileInputStream(p);
DataInputStream dis = new DataInputStream(fis);
System.out.println(dis.readInt());
System.out.println(dis.readDouble());
dis.close();
fis.close();
}
}
```

Concatenating and Buffering Files

Concatenation is achieved by using SequenceInputStream class . and Buffering is done by using BufferedInputStream and BufferedOutputStream.

class buf

```
{
    public static void main(String args [ ]){
        FileInputStream f1 = new FileInputStream( “test1.java”);
        FileInputStream f2 = new FileInputStream( “test2.java”);
file3 = new SequenceInputStream (f1,f2);
BufferedInputStream inb  = new BufferedInputStream(file3);
BufferedOutputStream outb  = new BufferedOutputStream(System.out);

while((ch=inb.read())!=-1)
{
outb.write((char)ch);
}
inb.close();
oub.close();
f1.close();
2.close();
}
}andom Access File
```

RandomAccessfile class supported by the Java.io package allows us to create files that can be used for reading and writing data with random access

```
class ran
{
public static void main(String args[])
{
RandomAccessFile  f = new RandomAccessFile(“rand.dat”,”rw”);
f.WriteChar(‘X’);
f.WriteInt(44);
file.seek(0);
System.out.println(f.readChar());
System.out.println(f.readInt());
f.seek(2);
```

```
System.out.println(f.readChar());
}}
```

Suggested Questions

1. **How to create and manipulate the files.**
2. **What is the usage of Random access files and explain it.**
3. **Explain FileInputStream and FileOutputStream.**
4. **Explain Stream class.**
5. **How to read and write bytes by using files.**