

VIDYAVAHINI FIRST GRADE COLLEGE



JAVA PROGRAMMING

5TH Sem BCA

Vidyavahini First Grade College

Near Puttanjaneya Temple, Kuvempunagar, Tumkur – 572103.

E-Mail: vvfgc.bca@gmail.com

Website: www.vidyavahini.org/bca

Contact No: 0816 – 2261130

Chapter 1

Introduction to JAVA

History of JAVA

- **Java** is a [general-purpose](#), [concurrent](#), [class-based](#), [object-oriented](#) [computer programming language](#) developed by the Sun Microsystems.
- Earlier, C++ was widely used to write object oriented programming languages; however, it was not a platform independent and needed to be recompiled for each different processor.
- Whereas Java applications are typically [compiled](#) to [byte code](#) ([.class file](#)) that can run on any platform (OS + Processor).
- It is intended to let application developers "**write once, run anywhere**" (WORA), meaning that *code that runs on one platform does not need to be recompiled to run on another*.
- [James Gosling](#), Mike Sheridan, and [Patrick Naughton](#) initiated the Java language project in **June 1991**.
- The language was initially called [Oak](#) after an [oak](#) tree that stood outside Gosling's office.
- It went by the name *Green* later, and was later renamed *Java*, from [Java coffee](#), said to be consumed in large quantities by the Java language's creators.

JAVA Versions

Major release versions of Java, along with their release dates:

- ✓ JDK 1.0 (January 21, 1996)
- ✓ JDK 1.1 (February 19, 1997)
- ✓ J2SE 1.2 (December 8, 1998)
- ✓ J2SE 1.3 (May 8, 2000)
- ✓ J2SE 1.4 (February 6, 2002)
- ✓ J2SE 5.0 (September 30, 2004)
- ✓ Java SE 6 (December 11, 2006)
- ✓ Java SE 7 (July 28, 2011)
- ✓ Java SE 8 (March 18, 2014)

JAVA Editions

Java Acronym	Java Edition	Formerly called	Use
Java SE	Java <i>Standard</i> Edition	J2SE	It contains basic core java classes. It is used to develop standard applets and applications Ex: <i>Developing stand alone payroll application for small company.</i>
Java EE	Java <i>Enterprise</i> Edition	J2EE	To develop server-side applications such as Java servlets and Java Server Pages. Ex: <i>Developing Internet Banking Applications.</i>
Java ME	Java <i>Micro</i> Edition	J2ME	Used to develop applications for portable devices such as <ul style="list-style-type: none">➤ mobile (e.g., cell phone, PDA)➤ embedded devices (e.g., TV tuner box, printers) Ex: <i>Developing game for mobile devices.</i>

JAVA features**Simple**

Java Inherits the C/C++ syntax and many of the object oriented features C++.

Many confusing features like pointers and operator overloading are removed from java.

So for the professional programmers learning java is easy.

Java Is Object-Oriented

Java is a true object oriented language which supports all the features of OOP like Inheritance, Encapsulation and Polymorphism. Almost everything is an object in java. All program code and data reside within classes.

Java Is Distributed

Java is designed for the distributed environment of the internet, because it handles TCP/IP protocols like HTTP and FTP.

Java supports network programming to communicate with remote objects distributed over the network. Java provides libraries like RMI and CORBA to develop network applications.

Architectural Neutral

The java Compiler generates byte code, which has nothing to do with particular computer architecture; hence a Java program is easy to interpret on any machine.

Portable

Java Byte code can be carried to any platform. No implementation dependent features. Everything related to storage is predefined, example: size of primitive data types.

Robust

Robust simply means strong. Java uses strong memory management. There is automatic garbage collection in java. There is exception handling and type checking mechanism in java which makes java robust.

Multithreaded

Java multithreading feature makes it possible to write program that can do many tasks simultaneously. Benefit of multithreading is that it utilizes same memory and other resources to execute multiple threads at the same time.

Ex: While typing in MS Word, grammatical errors are checked along.

Platform Independent

Java programs can be run on computer systems with different OS and processor environment. Java code is compiled by the compiler and converted into byte code. This byte code is a platform independent code because it can be run on multiple platforms.

Secure

Java security feature enable us to develop virus free applications. Java programs always run in Java runtime environment with null interaction with system OS that restricts them from introducing virus, deleting or modifying files in the host computers, hence it is more secure.

Compiled and interpreted language

Java uses a two step translation process. Java source code is compiled down to "byte code" by the Java compiler (javac). Then byte code is converted into machine code by the Java Interpreter (Java Virtual Machine - JVM).

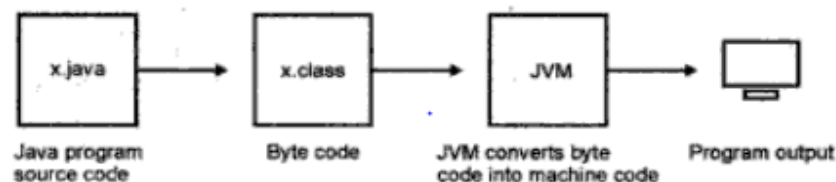


Figure 1.5 Execution of a Java program

Differences between C and JAVA

SL.NO.	C	JAVA
1	C is procedure oriented programming language.	JAVA is Object oriented programming Language
2	Pointers concept is used	Pointers concept is not used
3	C has goto statement	No goto statement
4	#define,typedef and header files are available in C	#define,typedef and header files are not available in JAVA
5	Supports sizeof operator	JAVA does not supports sizeof operator
6	Supports structures and unions	JAVA does not supports structures and unions
7	Manual memory management in C	Automatic memory management in JAVA
8	C supports storage classes and signed,unsigned modifiers	C supports storage classes and signed,unsigned modifiers

Differences between C++ and JAVA

SL.NO.	C++	JAVA
1	Pointers concept is used	Pointers concept is not used
2	Operator overloading is available in C++	No Operator overloading in java.
3	There are 3 access specifiers in C++: private, public, protected.	There are 4 access specifiers in JAVA: default, private, public, protected.
4	Supports Multiple inheritance	Does not supports multiple inheritance
5	#define,typedef and header files are available in C	#define,typedef and header files are not available in JAVA
6	Supports sizeof operator	JAVA does not supports sizeof operator
7	Supports template classes	JAVA does not supports template classes
8	Manual memory management in C	Automatic memory management in JAVA
9	There are constructors and destructors in C++.	Only constructors are available in JAVA. No destructors.

Java Development Kit (JDK)

It is a program development environment for developing java applets and applications.

It includes JRE and command line development tools.

When installed on a computer, the JRE provides the operating system with the means to run Java programs

It provides the **libraries**, the Java **Virtual Machine**, and other components to run applets and applications written in the Java programming language.

JDK Components: JDK includes the following command line development tools

Tool Name	Purpose
javac	The compiler for the java programming language. Using this tool we can compile java programs and generates byte codes.
Java	The interpreter for java applications used to execute the java byte codes.
javadoc	API documentation generator used to generate documentation for our java programs.
appletviewer	Used to run and debug applets without a web browser.
Jar	Create and manage java archive.
Jdb	Java debugger used to debug our java programs to find out any errors.
Javah	C header and stub generator.Used to write native methods.
javap	Java disassemble used to convert byte codes to a program description

Java API (Application Programming Interface)

The Java APIs are pre written Java code that can be used by other programmers to create Java applications. The java API is the set of huge number of classes and methods grouped into packages and it is included with the java development environment.

The Most commonly used Java API packages Are:

Sl.No.	Package name	Purpose	Description
1	java.lang	Language support	This package contains classes that support basic language features and handling arrays and strings.
2	java.io	Input/output	This package contains classes for input and output operations
3	java.util	Utilities	This package contains various utility classes like date and time and managing data within collections.
4	java.applet	Applets	This package contains classes used to create java applet programs.
5	java.awt	Abstract window toolkit	This package contains classes that allow us to write GUI programs.

Integrated Development Environment (IDE)

It is a software application that provides comprehensive facilities to computer programmers for software development.

An IDE normally consists of:

- a source code editor
- a compiler and/or an interpreter
- build automation tools
- a debugger

Popular Java IDEs are:

- Net Beans Open Source by Sun
- Eclipse Open Source by IBM

What is Byte code?

Byte code is the code produced by java compiler after compiling the Java source code. The byte code is stored in a file with .class extension. These byte code instructions are cannot be understand by processor. They are understandable only by JVM.

What makes Java as platform independent language?

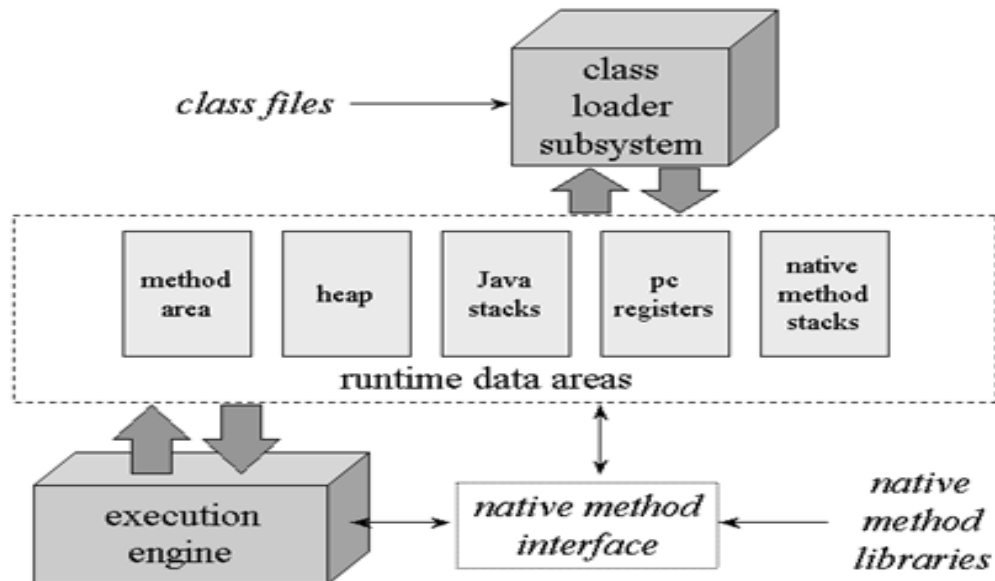
- In java if we write a program, it is stored with an extension .java say Hello.java which contains java source code statements.
- Then the source code is compiled into byte code using javac compiler which generates a file with .class extension say Hello.class.
- The instructions of byte code are understandable by JVM, which is program written to convert the byte code into machine understandable code.
- So the processor executes the converted byte code and displays the results.

In this way we can execute the .class file on any computer system with any processor and any operating system –provided JVM is available.

Thus java is called platform independent language.

Java Virtual Machine

- It is a program written to understand the byte code instructions and convert them into machine code.
- The role of JVM is to identify the processor and operating system used in the computer system and **converts the byte code instructions in an understandable format for the particular processor and operating system.**

Architecture of JVM (components of JVM)**Class Loader Sub System:**

- The class loader sub system is responsible for loading the .class file (Byte code) into the JVM.
- Before loading the Byte code into the JVM it will verify where there the Byte code is valid or not. This verification will be done by **Byte code verifier**.
- If the Byte code is valid then the **memory for the Byte code will be allocated in different areas**.
- The different areas into which the Byte code is loaded are called as Run Time Data Areas. The various run time data areas are:

1. Method Area:

- This Area can be used for storing all the class code and method code.
- All classes' byte code is loaded and stored in this run time area, and all static variables are created in this area.

2. Heap Memory:

- This Area can be used for storing all the **objects** that are created.
- The JVM allocates memory in heap for and object which is created using 'new' operator.

3. Java Stack area:

- This Area can be used for storing the information of the methods. That is under execution.
- In this runtime area all Java methods are executed.

4. PC Register (program counter) area:

- This Register will contain address of the next instruction that have to be executed.

5. Java Native Stack:

- This area is used for storing non-java coding available in the application. The non-java code is called as native code.

Execution Engine:

- The Execution Engine is Responsible for executing the program and it contains two parts.
 - ✓ Interpreter.
 - ✓ JIT Compiler (just in time compiler).
- The java code will be executed by both interpreter and JIT compiler simultaneously which will reduce the execution time and then by providing high performance. The code that is executed by JIT compiler is called as **HOTSPOTS** (company).

What is JIT?

The current version of Sun Hotspot JVM uses a technique called [Just-in-time \(JIT\) compilation](#) to compile the byte code to the native instructions understood by the CPU on the fly at run time.

Structure of Java Program

Documentation section
Package Statement
Import Statement
Interface Statement
Class Statement
main method class { main method definition }

- a. **Documentation section:** A set of comment lines like: name of the program, author name, date of creation, etc.
- b. **Package statement:** This is the first statement allowed in Java file, this statement declares a package names.
- c. **Import statement:** This statement imports the packages that the classes and methods of particular package can be used in the current program.
- d. **Interface statement:** An interface is like a class but includes group of methods declaration. Inter face is used to implement the multiple inheritance in the program.
- e. **Class definition/statements:** A Java program may contains one more class definitions.
- f. **Main method class:** Every Java standalone program requires a main method, as it begins the execution, this is essential part of Java program

Simple Java Program

```
import java.io.*;
public class Welcome
{
    public static void main (String args[])
    {
        System.out.println ("welcome to Java Program");
    }
}
```

public: It is access specifier and this method can be called from outside also.

static: This main method must always declared as static because the whole program has only one main method and without using object the main method can be executed by using the class name.

void: The main method does not return anything.

main: - The main method similar to main function in c and c++. This main method call first (execute first) by the interpreter to run the java program.

String: It is built-in class under language package.

args []: This is an array of type string. a command line arguments holds the argument in this array.

System.out.println:

System: It is a class which contains several useful methods.

out: It is an object for a system class to execute the method.

println: This is a method to print string which can be given within the double coats. After printed the contents the cursor should be positioned at beginning of next line

print: after printed the content the cursor should be positioned next to the printed content

To compile and run simple program

- ❖ Type the source code in any text editor like notepad
- ❖ Save the title name as class name java in the bin folder[welcom.java]
- ❖ To compile the source code file Welocome.java
 - Open a command prompt and move to the directory
 - C:\Program Files\Java\jdk1.6\bin>
 - **javac** is used to compile the source code

C:\program Files\java\jdk1.6\bin>java Welcom.java

It successfully compiled with no-errors, Java compiler converts source code [welcome.java] to byte code [welcome. class] into particular machine instruction to run java program

To execute the Java program outside the bin folder

- ❖ Save java file in D drive
- ❖ Set the path where bin\folder path
- ❖ D:\>set path=c:\program files\java\jdk1.6\bin
- ❖ D:\>javac Welcome.java
- ❖ D:>java Welcome

OOPs (Object Oriented Programming System)

Object means a real word entity such as pen, chair, table etc. Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Object

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be tangible or intangible entity.

Class

Collection of similar type objects is called class.

Ex: Apple, Mango, orange are the objects of class FRUITS.

Inheritance

When one object acquires all the properties and behaviors of parent object i.e. known as inheritance. It provides extensibility.

Polymorphism

Same name multiple forms is known as polymorphism.

In java, we use method overloading and method overriding to achieve polymorphism.

Information Hiding

Controlling access to the information provided to the user using access specifies.

For example: phone call, we don't know the internal processing.

Encapsulation

Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

JAVA programming Fundamentals

Java Tokens: Tokens are the basic building-blocks of the java language.

There exist 6 types of tokens in JAVA.

1. Keywords
2. Identifier
3. Literals
4. Separators
5. Operators
6. Comments

Keywords: The Java programming language has total of 50 reserved keywords which have special meaning for the compiler and cannot be used as variable names. Following is a list of Java keywords in alphabetical order, click on an individual keyword to see its description and usage example.

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

Some noteworthy points regarding Java keywords:

- **const** and **goto** are reserved words but not used.
- **true**, **false** and **null** are literals, not keywords.
- All keywords are in lower-case.

Identifier: Identifier is case sensitive names given to variable, methods, class, object, packages etc

Rules for identifiers:

- 1) May consists of letters, digits, underscore and a dollar symbol.
- 2) Must begin with a letter (A to Z or a to z), dollar (\$) or an underscore (_).
- 3) A key word cannot be used as an identifier.
- 4) They are case sensitive.
- 5) Whitespaces are not allowed.

Examples of legal identifiers: age, \$salary, _value, __1_value

Examples of illegal identifiers: 123abc, -salary

Literals: Literals used to indicate constant **values** in Java program.
There are 5 types of literals in java.

Integer literals:

Integer data that can be expressed in decimal (base 10), hexadecimal (base 16) or octal (base 8) number systems as well.

Prefix **0** is used to indicate octal and prefix **0x** indicates hexadecimal when using these number systems for literals.

Ex:

- 1) int decimal = 100;
- 2) int octal = 0144;
- 3) int hexa = 0x64;

Floating literals: Floating point data consist of float and double values.

Ex: 0.6D, 33.7F, 2.6F.....

Character literals: Any single character enclosed within single quotes.

Ex: 'A', 'B', 'D', '+', '/'...

Boolean literals: Represents Boolean true or false values.

Ex: true, false

String literals: Sequence of characters enclosed within double quotes.

Ex: "tumkur", "Vvfgc"

Separators: They are character used to group and arrange Java source code into segments. Java uses 6 types of separators:

Symbol	Name	Purpose
()	Parentheses	Used to establish precedence in an expression or to contain parameter list for methods.
{ }	Braces	Used to begin and end a block of code.
[]	Brackets	Used represent the index of an array.
;	Semicolon	Used to mark the end of statements.
,	Comma	Separates consecutive identifiers in a variable declaration.
.	Period	Used to separate package names from sub packages and classes.

Comments:

A comment is a non executable portion of a program that is used to document the program.

There are 3 types of comments in java:

- 1) Single line comment: //
- 2) Multiline comment: /* */
- 3) Documentation comment: /** */

Data types:

Java is strongly typed language. Data type specifies the type of value that can be stored in a variable.

Type	Size	Range
byte	8 bits	-128 to +127
short	2 bytes	-32768 to +32767
int	4 bytes	-2147483648 to +2147483647
float	4 bytes	-3.4×10^{38} to 3.4×10^{38}
char	2 bytes	0 to 65,535
double	8 bytes	1.7×10^{-308} to $1.7 \times 10^{+308}$
long	8 byte	-9223372036854775808 to 9223372036854775807
boolean	1 byte	True/false[default false]

Types of variables in Java or scope and lifetime of variable:

Scope determine when the variable is created and destroyed

Java has 3 types of variables:

1) Local Variables:

The variables declared in a method, constructor or block.

They are created when the method starts execution and destroyed when the method terminates execution.

Local variables are not initialized automatically.

2) Instance variables [Member variables]

The variables declared in a class but outside a method.

Instance variable is created when the object is created and destroyed when the object is destroyed.

Initialized to default values based on the data type used.

3) Static variables [class variables]

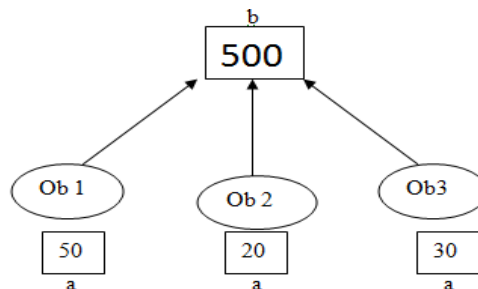
The variables declared in a class with the **static** keyword but outside a method.

There is only one copy per class regardless of how many objects are created from it.

The scope begins when the program execution starts and scope ends when the program execution stops.

Initialized to default values based on the data type used.

```
Ex:      class Scope
        {
            int a; //Instance variable
            static int b=500;    //Static variable or class variable
            void add()
            {
                int c; //Local variable
            }
            public static void main(String args[])
            {
                Scope Ob1, Ob2, Ob3;
            }
        }
```



❖ Here **b** is an static variable or class variable which is common to all object

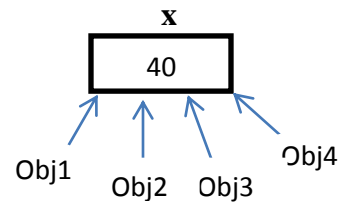
❖ **a** is an instance variable or member variable which is related to particular object

static keyword: static keyword can be used to create

1. Static variables
2. Static methods
3. Static block

Static variable: They are also called as class variables & static variable can be before any objects of its class are created.. Static variables creates single memory block & are common to the all object.

Ex: static int x;



A static variable can be accessed by directly by the class name & no need of an object.

Syntax: `<class name>. <Static_variable_name>;`

Static method: The methods declared as static are static methods.

The static method can be called without using objects.

static method can access only static data.

They cannot refer instance variables.

Syntax: `<class name>. <method name> (arguments);`

```
Ex: public class StaticDemo
{
    static int a=100,b=200;//static data
    static statMethod ()//static method
    {
        System.out.println("b:" +b);
    }

    public static void main(String args[])
    {
        System.out.println(StaticDemo.a);//static data accessed using classname
        StaticDemo.statMethod();//static method accessed using classname
    }
}
```

Static block: The static block is used to initialize all the static variables before using them in computation.

The static block gets executed exactly once when the class is first loaded.

Ex:

```
class StaticBlock
{
    static int a=7;
    static int b;
    void printValues()
    {
        System.out.println("a:"+a);
        System.out.println("b:"+b);
    }
    static // static block
    {
        System.out.println("Inside Static Block");

        b=a+5;
    }
    public static void main(String args[])
    {
        StaticBlock obj=new StaticBlock();
        obj.printvalues();
    }
}
```

Command Line arguments: The values passed in the command line to the program during execution after the file name are called command line arguments.

Command line arguments are stored in args[] array of main() method.

It is a way of providing input to the java program.

Ex: java Hello **welcome java**

These arguments will be stored in args[] array as follows:

```
args[0]= "welcome";
args[1]= "java";
```

Java program to accept and print command line arguments

```
import java.lang.*;
public class CmdLineArgsDemo
{
    public static void main(String args[])
    {
        System.out.println ("Command line arguments");
        System.out.println (args[0]);
        System.out.println (args[1]);
        System.out.println (args[2]);
    }
}
```

Control Statements

The control statements are used to control the flow of execution of the program.

Java contains the following types of control statements:

- 1) Selection Statements / Decision making statements: **if, if-else, switch.**
- 2) Repetition Statements / Looping Statements: **while, do-while, for.**
- 3) Branching Statements / Jumping Statements: **break, continue, and return.**

Selection statements:

if Statement:

This is a control statement to execute a single statement or a block of code, when the given condition is true and if it is false then it skips **if** block and rest code of program is executed .

Syntax: if (conditional expression) {
 <statements>;
}

Ex: If $n \% 2$ evaluates to 0 then the "if" block is executed. Here it evaluates to 0 so if block is executed. Hence **"This is even number"** is printed on the screen.

```
int n = 10;  
if (n%2 == 0){  
    System.out.println ("This is even number");  
}
```

if-else Statement:

The **"if-else"** statement is an extension of if statement that provides another option when 'if' statement evaluates to "false" i.e. else block is executed if **"if"** statement is false.

Syntax:

```
if(conditional expression){  
    <statements>;  
}  
else{  
    <statements>;  
}
```

Ex: If $n\%2$ doesn't evaluate to 0 then else block is executed. Here $n\%2$ evaluates to 1 that is not equal to 0 so else block is executed. So **"This is not even number"** is printed on the screen.

```
int n = 11;
if(n%2 == 0){
    System.out.println("This is even number");
}
else{
    System.out.println("This is not even number");
}
```

switch Statement:

The keyword **"switch"** is followed by an expression that should evaluate to byte, short, char or int primitive data types only. In a switch block there can be one or more labeled cases. The switch expression is matched with each case label. Only the matched case is executed, if no case matches then the default statement (if present) is executed.

Syntax:

```
switch(expression){
    case expression 1:<statement>;break;
    case expression 2:<statement>;break;
    ...
    case expression n: <statement>;break;
    default:<statement>;
} //end switch
```

Ex: Here expression "day" in switch statement evaluates to 5 which matches with a case labeled "5" so code in case 5 is executed that results to output **"Friday"** on the screen.

```
int day = 5;
switch (day) {
    case 1: System.out.println("Monday"); break;
    case 2: System.out.println("Tuesday"); break;
    case 3: System.out.println("Wednesday");break;
    case 4: System.out.println("Thrusday"); break;
    case 5: System.out.println("Friday"); break;
    case 6: System.out.println("Saturday");break;
    case 7: System.out.println("Sunday");break;
    default: System.out.println("Invalid entry");
}
```

Repetition Statements:**while:**

This is a looping or repeating statement. It executes a block of code or statements till the given condition is true.

Syntax:

```
while(expression){  
<statement>;  
}
```

Ex: Here expression $i \leq 10$ is the condition which is checked before entering into the loop statements. When i is greater than value 10 control comes out of loop and next statement is executed.

```
int i = 1;  
//print 1 to 10  
  
while (i <= 10){  
  
    System.out.println("Num " + i);  
  
    i++;  
  
}
```

do-while loop statements:

This is a post-test loop statement. First the **do** block statements are executed then the condition given in **while** statement is checked. So in this case, even the condition is false in the first attempt, do block of code is executed at least once.

Syntax:

```
do{  
<statement>;  
}while (expression);
```

Ex: Here first do block of code is executed and current value "1" is printed then the condition $i \leq 10$ is checked. Here "1" is less than number "10" so the control comes back to do block. This process continues till value of i becomes greater than 10.

```
int i = 1;  
do{  
  
    System.out.println("Num: " + i);  
  
    i++;  
  
}while(i <= 10);
```

for loop statements:

This is also a loop statement that provides a compact way to iterate over a range of values. it executes the statements within the block repeatedly till the specified conditions is true .

Syntax:

```
for (initialization; condition; increment or decrement){  
    <statement>;  
}
```

initialization: The loop is started with the value specified.

condition: It evaluates to either 'true' or 'false'. If it is false then the loop is terminated.

increment or decrement: After each iteration, value increments or decrements.

Ex: Here num is initialized to value "1", condition is checked whether num<=10. If it is so then control goes into the loop and current value of num is printed. Now num is incremented and checked again whether num<=10.If it is so then again it enters into the loop. This process continues till num>10. It prints values 1 to10 on the screen.

```
for (int num = 1; num <= 10; num++){  
    System.out.println("Num: " + num);  
}
```

Branching Statements:**Break statements:**

The break statement is used for breaking the execution of a loop (while, do-while and for). It also terminates the switch statements.

Syntax:

```
break; // breaks the innermost loop or switch statement.  
break label; // breaks the outermost loop in a series of nested loops.
```

Ex: When if statement evaluates to true it prints "data is found" and comes out of the loop and executes the statements just following the loop.

```
int num[] = {2,9,1,4,25,50};

int search = 4;

for (int i = 1; i < num.length; i++){

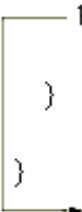
    if ( num[i] == search){

        System.out.println("data is found!");

        break;

    }

}
```

**Continue statements:**

This is a branching statement that are used in the looping statements (while, do-while and for) to skip the current iteration of the loop and resume the next iteration .

Syntax: continue;

Ex:

```
int num[] = {2,9,1,4,25,50};

int search = 4;

for (int i = 1; i < num.length; i++){

    if (num[i] != search){

        continue;

    }

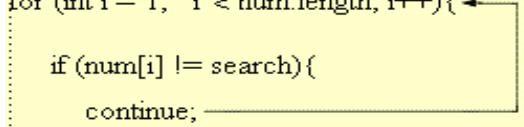
    if (Found == search){

        System.out.println("data is found!");

        break;

    }

}
```



return statements:

It is a special branching statement that transfers the control to the caller of the method. This statement is used to return a value to the caller method and terminates execution of method.

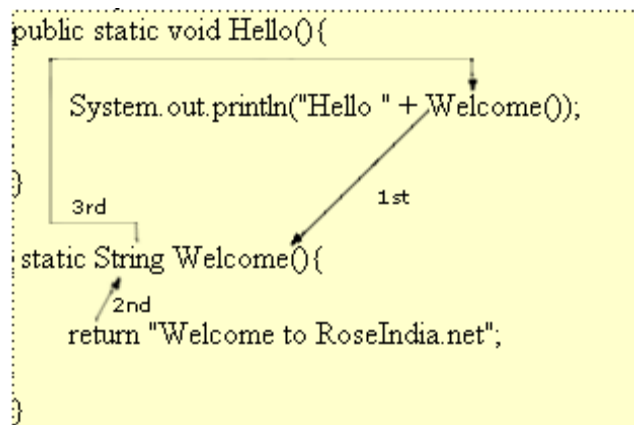
Syntax: return;

return values;

return; //This returns nothing. So this can be used when method is declared with void return type.

return expression; //It returns the value evaluated from the expression.

Ex: Here Welcome () function is called within println() function which returns a String value "Welcome to roseIndia.net". This is printed to the screen.

**Type casting or Data conversion**

Definition: Type casting is used to convert data from one data type to another data type

Integer conversion:

```
int a = Integer.parseInt(s1);
```

This is to convert the string s1 to integer data type using function parse int that can be called by class integer

Float Conversion:

```
float a = Float.parseFloat(s1);
```

This is to convert the string s1 to float data type using function parse float that can be called by class float

Double Conversion:

```
double a = Double.parseDouble(s1);
```

This is to convert the string s1 to double data type using function parse double that can be called by class double.

Reading Input from keyboard

Reading input from keyboard a stream is required to accept data from keyboard. A stream represents flow of data

```
DataInputStream ds = new DataInputStream (System.in);  
String s = ds.readLine();
```

System.in : We have created input stream object or DataInputStream object and connecting the keyboard system into it

readLine() : Used to accept or read the data in a line.

Write a program to accept user name and display the same

```
import java.io.*;  
public class InputDemo  
{  
    public static void main (String args[])throws IOException  
    {  
        DataInputStream ds = new DataInputStream(System.in);  
        System.out.println("Enter your Name");  
        String s= ds.readLine();  
        System.out.println("Your Name:" +s);  
    }  
}
```

Write a program to accept two numbers and print the addition of two numbers

```
import java.io.*;  
public class SumOfTwoNumbers  
{  
    public static void main (String args[]) throws IOException  
    {  
        int a, b, c;  
        DataInputStream ds = new DataInputStream(System.in)  
        System.out.println("Enter 2 numbers");  
        a=Integer.parseInt(ds.readLine());  
        b=Integer.parseInt(ds.readLine());  
        c=a+b;  
        System.out.println ("Sum="+c);  
    }  
}
```


Arrays

“An array is a collection of elements of the same data type”.

One-dimensional array:

Declaration: In JAVA an array declaration is a two step process:

- First, you should declare a variable by specifying its datatype

Syntax: **datatype[] variable_name;**

- Second, you should allocate the memory for an array using new and assign it to the array variable

Syntax: **variable_name =new datatype[size];**

The elements in the array allocated by *new* will *automatically be initialized to zero*.

Ex: 1) int a[];

2) a=new int[10];

The two steps can be combined into a single line as:

int a[]=new int[10];

Ex 1:

```
class ArrayDemo
{
    public static void main(String args[])
    {
        int a[]=new int[10];
        a[0]=10;
        a[1]=20;
        System.out.println(a[0]);
        System.out.println(a[1]);
    }
}
```

Ex 2:

```
class ArrayDemo1
{
    public static void main(String args[])
    {
        int a[]=new int[10];
        int count=10;
        for(int i=0;i<4;i++)
        {
            a[i]=count;
            count=count+10;
        }
        for(int i=0;i<4;i++)
        {
            System.out.println(a[i]);
        }
    }
}
```

Multidimensional array:

Multidimensional arrays are declared by specifying another set of subscript operator for each index.

For example two-dimesional array can be declared as follows

Syntax: datatype array_name[][];
array_name = new datatype [size][size];

Ex: int a[][]= new int [2][2];

Questions:

1. What is Domain Name Service? (Nov-2010:1-marks) (Nov-2012:1-marks)
2. Write the use of Web server? (Nov-2012:1-marks)
3. Who developed Java & when?
4. Explain the features of JAVA.
5. What is instance of operator? (Nov-2010:1-marks)
6. What is the role of JVM & explain its components? (Nov-2012:7-marks)
7. Write the difference between Java & c++. (Nov-2010:3-marks)
8. Explain JDK components.
9. Explain public static void main(String args[]). (Nov-2010:3-marks)
10. How Java is platform independent? (Nov-2010:1-marks)
11. Explain Java API packages.
12. What are the types of Java program.
13. What are the different phases of Java program development? (Nov-2012:3-marks)
14. Explain the structure of Java program.
15. What are the disadvantages of Java.
16. What are Java Tokens. (Nov-2012:1-marks)
17. What is typecasting?
18. Why java is Architecture neutral language ? Justify. (Nov-2012:3-marks)
19. Explain scope of a variable.
20. Explain conditional operator in Java with an example. (Nov-2010:2-marks)
21. Explain the creation of arrays.
22. Explain the operators in Java. (Nov-2012:4-marks)
23. Write a java program to find the factorial of a given integer number. (Nov-2012:3-marks)
24. What are the Data types in Java. (Nov-2011:1-marks) (Nov-2012:3-marks)
25. Write a program to print the multiplication table for a given number. (Nov-2010:5-marks)

Chapter-2

Class & Objects

Class: “A class is a collection both data member & member functions”

Or

“It is a collection of similar type of object”

Object: object is an instance of a class used to access the both Data member & member functions outside the class also.

Defining a class:

Syntax:

```
<access_specifier> class <class_name><extends><super_class><implements><interface_list>
{
    //data member
    //member function
}
```

Rules for defining a class:

- The name start with a capital letter
- They can be only one public class per program
- A program can have any number of non-public classes

Note: If we declare a class as private then it is not visible to java compiler & hence we get compile time error but inner class can be declared as private.

Ex: public class D extends A implements B

```
    {
        Int b;
        Void display()
        {
            .....
            .....
        }
    }
```

Adding methods to a class :

Syntax: return_type function_name(parameter_list)

```
{
    -----
    -----
}
```

Adding variable to a class :

The variable inside a class are of two types

1. Static variable or Class variable
2. Instance variable

Ex: class XYZ

```
{
    Int a;
    Static int b;
    void display()
    {
        -----
        -----
    }
}
```

Note: In java no classes are terminated in semicolon.

Creating object:

Syntax: <class_name> <object_name>=new <class_name>(arguments_list)

An object is created by instantiating a class. The process creating an object is called as instantiation and the created object is an instance.

The object is created by using ‘new’ operator

Ex: sum s=new sum();

Accessing Class member:

We can access the member of the class using Dot (.) operator.

Syntax: Obj_name.variabele_name;
 Obj_name.function_name();

Ex: s.a;
 s.add();

Write a program to accept two numbers & calculating its sum using Class & Object.

```
class Sum
{
    int a,b,ans;
    DataInputStream ds=new DataInputStream(System.in);
    void accept()
    {
        System.out.println("Enter the two Numbers");
        a=Integer.parseInt(ds.readLine());
        b=Integer.parseInt(ds.readLine());
    }
    void cal()
    {
        ans=a+b;
    }
}
```

```
        void display()
        {
            System.out.println("Ans="+ans);
        }
    }

    public class Ex
    {
        public static void main(string args[])
        {
            Sum S=new Sum();
            S.accept();
            S.cal();
            S.display();
        }
    }
```

Constructors:

They are used to initialize the object, constructor having the same name as the class name and executes automatically when the object is created.

Ex: class XYZ

```
{
    int x;
    XYZ()           //constructor
    {
        x=10;
    }
}
```

Types of constructor :

Constructors are three types:

1. Default constructor (Non-parameterized constructor)
2. Parameterized constructor
3. Constructor overloading

1. Default constructor:-

Default constructor having no parameters & Executed automatically when the object is created.

If we doesn't give the default constructor in the class, java automatically provides default constructors that initializes the variables as '0' or 'NULL'.

Ex: class XYZ

```
{
    int a,b;
    XYZ()           //default constructor or non-parameterised constructor
    {
        a=10;
        b=20;
    }
}
```

```
    }  
    }  
    public class Ex  
    {  
        public static void main(String args[])  
        {  
            XYZ obj=new XYZ();  
        }  
    }  
}
```

2. Parameterized constructor :-

The constructor having one or more parameters that is used to initialize the variable is called parameterized constructor.

Ex: class XYZ

```
{  
    int a,b;  
    XYZ(int x, int y)    //parameterised constructor  
    {  
        a=x;  
        b=y;  
    }  
}  
public class Ex  
{  
    public static void main(String args[])  
    {  
        XYZ obj=new XYZ(10,20);  
    }  
}
```

3. Overloaded constructor:

Two or more constructors having different parameters is called overloaded constructor.

Ex:

```
class XYZ  
{  
    int a,b,c;  
    XYZ()  
    {  
        a=1;  
        b=2;  
        c=3;  
    }  
    XYZ(int x, int y)  
    {  
        a=x;  
    }  
}
```

```
        b=y;
        c=20;
    }
    XYZ(int x, int y, int z)
    {
        a=x;
        b=y;
        c=z;
    }
}
public class Ex
{
    public static void main(String args[])
    {
        XYZ obj1=new XYZ();
        XYZ obj2=new XYZ(10,20);
        XYZ obj3=new XYZ(10,20,30);
    }
}
```

Write program to find the area of different shapes

```
class cube
{
    int l,b,h;
    cube()
    {
        l=1;
        b=1;
        h=1;
    }
    cube(int x)
    {
        l=x;
        b=2;
        h=2;
    }
    cube(int x,int y,int z)
    {
        l=x;
        b=y;
        h=z;
    }
    void display()
    {
        System.out.println("Volume:"+l*b*h);
    }
}
```



```
}
public class Lab4
{
    public static void main(String args[])
    {
        cube obj1=new cube();
        cube obj2=new cube(10);
        cube obj3=new cube(10,20,30);
        obj1.display();
        obj2.display();
        obj3.display();
    }
}
```

Method overloading:

Method names are same but parameters are different is called as method overloading.

Ex:

```
class Ex
{
    void add()
    {
        .....
    }
    void add(int x,int y)
    {
        .....
    }
    void add(int x,int y,int z)
    {
        .....
    }
}
```

Note: method overloading is one of the way to implement polymorphism in Java.

Write a program to find the area of different shapes using method overloading

```
class Area
{
    DataInputStream ds=new DataInputStream(System.in);
    void circle()throws IOException
    {
        double r,pi=3.142;
        System.out.println("Enter the radius");
        r=Double.parseDouble(ds.readLine());
        System.out.println("Area of circle="+(pi*r*r));
    }
    void square()throws IOException
```

```
        {
            int s;
            System.out.println("Enter the Sides of a SQUARE");
            s=Integer.parseInt(ds.readLine());
            System.out.println("Area of Square="+s*s);
        }
        void rect()throws IOException
        {
            int b,l;
            System.out.println("Enter the Bredth and Length");
            b=Integer.parseInt(ds.readLine());
            l=Integer.parseInt(ds.readLine());
            System.out.println("Area of Rectangle="+b*l);
        }
        void triangle()throws IOException
        {
            DataInputStream ds=new DataInputStream(System.in);
            int br,h;
            System.out.println("Enter the Bredth and Length");
            br=Integer.parseInt(ds.readLine());
            h=Integer.parseInt(ds.readLine());
            System.out.println("Area of Trangle =" + (0.5*br*h));
        }
    }
    public class Lab5
    {
        public static void main(String args[])throws IOException
        {
            Area obj=new area();
            obj.circle();
            obj.rect();
            obj.square();
            obj.triangle();
        }
    }
```

‘this’ Keyword

"this keyword refers to the current object". It always points object that is currently executing.

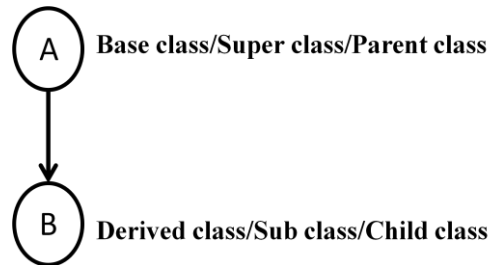
Ex:

```
class Xyz
{
    int x,y;
    {
        this.x=a;
        this.y=b;
    }
    void display()
    {
        System.out.println("X="+x);
        System.out.println("Y="+y);
    }
}

public class Ex
{
    public static void main(String args[])
    {
        Xyz obj1=new Xyz(10,20);
        Xyz obj2=new Xyz(100,200);
        obj1.display();
        obj2.display();
    }
}
```

Inheritance

Definition: “Deriving a new class from an existing class or acquiring the properties of super object from the sub class object”.



Concepts of Inheritance:

- The derived class members can access all the features of base class.
- Code reusability is one of the most powerful features of inheritance.
- Reusing of code saves time, money and effort.

Deriving sub class:

Syntax:

```
class <subclass_name> extends <superclass_name>
{
    -----/data member
    -----/member function
}
```

We can derive a sub class from an existing class using **Extends**.

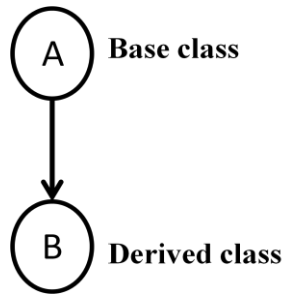
Types of inheritance:-

1. Single inheritance
2. Multilevel inheritance
3. Hierarchical inheritance
4. Multiple inheritance

1. **Single inheritance:** One derived class derives to one base class is known as single inheritance.

Or

One base class derives one derived class is known as single inheritance.

**Example:**

```
class A
{
    -----
}
class B extends A
{
    -----
}
```

Write a program to accept regno & name in base class & accept total marks & percentage in derived class and display the same.

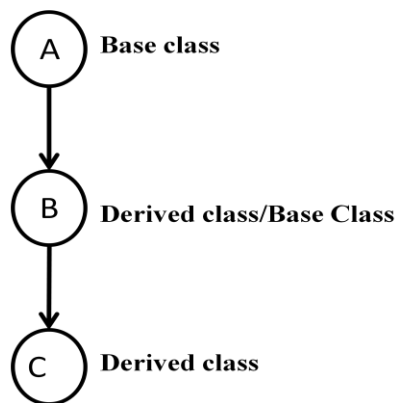
```
class A
{
    String name;
    int regno;
    void accept ()
    {
        DataInputStream ds=new DataInputStream(System.in);
        System.out.println("enter the name");
        name=ds.readLine();
        System.out.println("enter the Register number");
        regno=Integer.parseInt(ds.readLine());
    }
    void display()
    {
        System.out.println("Name="+name);
        System.out.println("Register number="+regno);
    }
}
class B extends class A
{
    void accept2()
    {
```

```
        int m1,m2,m3,total,per;
        System.out.println("enter the 3 subject marks ");
        m1=Integer.parseInt(ds.readLine());
        m2=Integer.parseInt(ds.readLine());
        m3=Integer.parseInt(ds.readLine());
    }

    void display2()
    {
        System.out.println("Toal="+m1+m2+m3);
        System.out.println("Average="+((m1+m2+m3)/3));
    }
}
public class Ex
{
    public static void main(String args[])
    {
        B obj=new B();
        obj.accept();
        obj.accept2();
        obj.display();
        obj.display2();
    }
}
```

Multilevel inheritance:-

A class can be derived from another derived class is known as multilevel inheritance.

**Example:**

```
class A
{
    -----
}
class B extends A
```

```
{
    -----
}
class C extends B
{
    -----
}
```

Write a program to accept regno & name in base class & accept total marks & percentage in derived class and display the same.

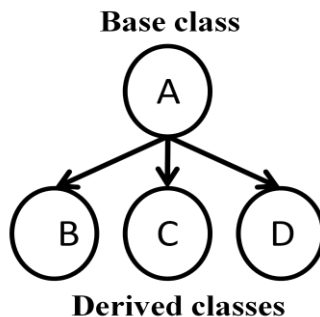
```
class A
{
    String name;
    int regno;
    void accept()
    {
        DataInputStream ds=new DataInputStream(System.in);
        System.out.println("enter the name");
        name=ds.readLine();
        System.out.println("enter the Register number");
        regno=Integer.parseInt(ds.readLine());
    }
    void display()
    {
        System.out.println("Name="+name);
        System.out.println("Register number="+regno);
    }
}
class B extends A
{
    void accept2()
    {
        String course, college;
        System.out.println("enter Your course ");
        course=ds.readLine();
        college=ds.readLine();
    }
    void display2()
    {
        System.out.println("Collage="+college);
        System.out.println("Course="+course);
    }
}
```

```
class C extends B
{
    void accept3()
    {
        int m1,m2,m3,total,per;
        System.out.println("enter the 3 subject marks ");
        m1=Integer.parseInt(ds.readLine());
        m2=Integer.parseInt(ds.readLine());
        m3=Integer.parseInt(ds.readLine());
    }
    void display3()
    {
        System.out.println("Toal="+m1+m2+m3);
        System.out.println("Average="+((m1+m2+m3)/3));
    }
}

public class Ex
{
    public static void main(String args[])
    {
        C obj=new C();
        obj.accept();
        obj.accept2();
        obj.accept3();
        obj.display();
        obj.display2();
        obj.display3();
    }
}
```

Hierarchical inheritance:-

One base class derives from two are more derived classes is called as hierarchical inheritance.



Example:

```
class A
{
    -----
}
class B extends A
{
    -----
}
class C extends A
{
    -----
}
class D extends A
{
    -----
}
```

write a program to accept regno & name in base class & accept total marks & percentage in derived class and display the same.

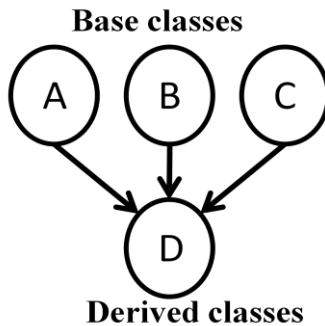
```
class course
{
    String name,regno;
    void accept()
    {
        DataInputStream ds=new DataInputStream(System.in);
        System.out.println("enter the name");
        Name=ds.readLine();
        System.out.println("enter the Register number");
        regno=ds.readLine();
        System.out.println("Name="+name);
        System.out.println("Register number="+regno);
    }
}
class bca extends course
{
    void accept2()
    {
        Int total,per;
        System.out.println("enter total marks ");
        total=Integer.parseInt(ds.readLine());
        System.out.println("enter average ");
        per=Integer.parseInt(ds.readLine());
    }
}
```

```
        System.out.println("Toal="+total);
        System.out.println("Average="+averag);
    }
}
class bsc extends course
{
    void accept3()
    {
        int total,per;
        System.out.println("enter total marks ");
        total=Integer.parseInt(ds.readLine());
        System.out.println("enter average ");
        per=Integer.parseInt(ds.readLine());
        System.out.println("Toal="+total);
        System.out.println("Average="+averag);
    }
}
class bcom extends course
{
    void accept4()
    {
        int total,per;
        System.out.println("enter total marks ");
        total=Integer.parseInt(ds.readLine());
        System.out.println("enter average ");
        per=Integer.parseInt(ds.readLine());
        System.out.println("Toal="+total);
        System.out.println("Average="+averag);
    }
}

public class Ex
{
    public static void main(String args[])
    {
        bca obj=new bca();
        bsc obj2=new bsc();
        bcom obj3=new bcom();
        obj.accept();
        obj2.accept2();
        obj2.accept3();
        obj3.accept4();
    }
}
```

Multiple inheritance:-

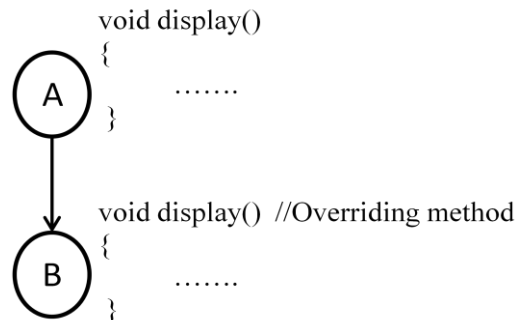
A derived class having two or more base classes is known as **multiple inheritances**”



Java does not supports multiple inheritance, it will be implemented by “**Interface concept**”.

Method overriding

The return type, argument list, method names are same in both base class and derived class is called as **method overriding**.



We cannot call the base class method from the derived class object.

Ex:

```
class A
{
    void display()
    {
        System.out.println("Class A");
    }
}
class B extends A
{
    void display() //Overriding Method
    {
        System.out.println("Class B");
    }
}
```

```
public class Ex
{
    public static void main(String args[])
    {
        B obj=new B();
        Obj.display();
        Obj.display();
    }
}
```

Super Keyword:-

The super keyword refers to the super class object.

If we want to call super class contractor explicitly, then the keyword super in java is used.

Super keyword is also used to access data member and member function of the super class that has been hidden by a member of a sub class.

Uses of “super” keyword:-

- It is used to call the super class constructor explicitly.
- It is used to access the super class variables inside the sub class method.
- It is used to access the super class methods from sub class method.

Rules for using Super keyword:-

- Super should be the first statement inside the constructor.
- Super and this keyword cannot be together.
- Super cannot be used inside the static method.
- If we do not write super then java provides super by default.

Ex:

```
class A
{
    int x;
    A()
    {
        System.out.println("Class A");
    }
    A(int a)
    {
        x=a;
        System.out.println("Class A"+x);
    }
}
class B extends A
{
    int b;
    B()
    {
        System.out.println("Class B");
    }
}
```

```
    }  
    B(int y)  
    {  
        super(10);  
        b=y;  
        System.out.println("Class B"+y);  
    }  
}  
Class Ex  
{  
    public static void main(String args[])  
    {  
        B obj1=new B();  
        B obj2=new B(100);  
    }  
}
```

To call super variable:-**Ex:**

```
class A  
{  
    int x=10;  
    void display()  
    {  
        System.out.println("X="+x);  
    }  
}  
class B extends A  
{  
    int x=20;  
    void display()  
    {  
        System.out.println("X="+x);  
        System.out.println("X="+super.x);  
    }  
}  
public class Ex  
{  
    public static void main(String args[])  
    {  
        B obj=new B();  
        obj.display();  
    }  
}
```

To call super class method:-

Ex:

```
class A
{
    int x=10;
    void display()
    {
        System.out.println("Super class X="+x);
    }
}
class B extends A
{
    int x=20;
    void display()
    {
        super.display();
        System.out.println("XSub class =" +x);
    }
}
public class Ex
{
    public static void main(String args[])
    {
        B obj=new B();
        obj.display();
    }
}
```

‘final’ keyword:

The **final** keyword is used to create constant names(variables), to avoid overriding and to avoid instance technique.

final variable:

Declaring a variable with the final keyword makes that variable at constant once we design the value to final variable, we can't change it again.

Final variable are used to create Constant variable.

Ex:

```
class A
{
    final int x=10;        // x is a constant variable
    void display()
    {
        int x=20;        // not allowed
        System.out.println("X="+x);
    }
}
public class Ex
{
}
```

```
        public static void main(String args[])
        {
            A obj=new A();
            obj.display();
        }
    }
```

final method:

A method can be declared as final to prevent sub classes from overriding it. The final method can't be overridden.

Ex:

```
class A
{
    final void display()
    {
        System.out.println("HELLO");
    }
}
class B extends A
{
    void display() // not allowed
    {
        System.out.println("BYE");
    }
}
```

final class:- The keyword final can also be used for class if we declare a class as final then that class cannot be inherited.

The java has a lot of built-in final classes like String, Math, etc.

Ex:

```
final class A
{
    void display()
    {
        System.out.println("Hai");
    }
}
class B extends A // not allowed
{
    void display()
    {
        System.out.println("BYE");
    }
}
```

Advantages of final Keyword:-

- It is used to create constant variables.
- It is used to prevent method overriding.
- It is used to prevent inheritance.

Abstract class and Abstract method

The method doesn't have definition, It just contains only method declaration is called as abstract method.

An abstract class is class which contains '0' or more abstract methods.

Ex: abstract class XYZ
 {
 abstract void method1();
 abstract void method2();
 void method3()
 {

 }
 }

- An abstract class cannot be instantiated
- Any class that contains abstract methods must be an abstract class.
- An abstract method can be implemented in their sub classes.

Difference between Class & Abstract-class

Class	Abstract class
1. It doesn't contains abstract method.	1. It contains abstract method and also ordinary methods.
2. Class can be instantiated.	2. Abstract class cannot be instantiated.

Ex:
abstract class College
{
 abstract void accept();
 abstract void display();
}
class Bca extends College
{
 void accept()
 {

 }


```
    }
    void display()
    {
        -----
        -----
    }
}
class Bsc extends College
{
    void accept()
    {
        -----
        -----
    }
    void display()
    {
        -----
        -----
    }
}
```

Garbage collection:

Java automatically release the allocated memory is called as the automatic garbage collection. Every object created for the particular class, there is a memory is allocated to that objects, if there is no use of object references then it will canceled to be garbage. The java releases the memory automatically.

JVM is responsible for realizing the memory allocated to a garbage collection.

Finalize method: we can do some cleanup operations this operation is known as finalization.

An object is garbage collected the memory will be cleanup using the finalize method & it reuse for the further operation or memory allocations. This method is a member of the java.lang.object class.

```
class XYZ
{
    protected void finalize() throws IOException
    {
        System.out.println("Garbage collection object");
    }
}
```

Write a program to calculate salary of different department using abstract class

```
abstract class dept
{
    double salary,bonus,total;
    abstract void cal(double sal);
    void display(String dept)
```

```
        {
            System.out.println(dept+" "+salary+" "+bonus+" "+total);
        }
    }
class accounts extends dept
{
    void cal(double sal)
    {
        salary=sal;
        bonus=salary*0.5;
        total=salary+bonus;
    }
}
class sales extends dept
{
    void cal(double sal)
    {
        salary=sal;
        bonus=salary*0.7;
        total=salary+bonus;
    }
}
class mange extends dept
{
    void cal(double sal)
    {
        salary=sal;
        bonus=salary*0.9;
        total=salary+bonus;
    }
}
public class Ex
{
    public static void main(String args[])
    {
        accounts nn1=new accounts();
        sales nn2=new sales();
        mange nn3=new mange();
        nn1.cal(10000);
        nn2.cal(12000);
        nn3.cal(15000);
        nn1.display("accounts");
        nn2.display("sales");
        nn3.display("mange");
    }
}
```

Strings

A combination or collection of characters is called as **string**.

The strings are objects in java of the class 'String'.

```
System.out.println("Welcome to java");
```

The string "welcome to java" is automatically converted into a string object by java.

A string in java is not a character array and it is not terminated with "NULL".

String are declared and created:-

Using character strings

```
String str1=new String("vvfgc");
```

```
String str2=new String(str1);
```

```
String str3="Tumkur"
```

We can also create string object by assigning value directly.

String object also create by using either new operator or enclosing of characters in double codes.

Java string can be con-catenation using the plus operator (+).

```
String name1="vvfgc";
```

```
String name2="college";
```

```
String str1=name1+name2;
```

```
String str2="Narasimha"+"murthy";
```

```
String str3=name1+"murthy";
```

```
String str4="Narasimha"+name2;
```

Methods of string classes :-

1. Length:

```
public int length();
```

It will give the length of a string.

Ex:

```
String str="murthy"
```

```
int n=str.length();
```

Here the length function returns the value seven.

```
System.out.prntln(str.length)//it is also returns the value 7
```

2. concat:

```
public String concat(String str);
```

It used to concating of two string.

Ex: String str1="murthy"
String str2="college"
String str3=str1.concat(str2);
'+' operator is also used to concatenate of two string.

3. equals:

public Boolean equals(String obj);

This method also check weather two strings are equal or not . it returns **true** if the two strings are equal otherwise it returns **false**.

Ex: String str1="college"
String str2="college"
if (str1.equals(str2))
 System.out.println("two strings are equal");
else
 System.out.println("two strings are Not equal");

4. equalsIgnoreCase:-

public Boolean equalsIgnoreCase(String obj);

The method ignores the case while comparing the content, It return **True** when the two strings character are in the different cases.

Ex: String str1="college"
String str2="college"
Str1.equalsIgnoreCase(str2);

5. toLowerCase:-

public String toLowerCase();

This method converts all the character to Lower case.

Ex: String str1="WELcome TO java"
String str2=str1.toLowerCase();

6. toUpperCase:-

public String toUpperCase();

This method converts all the character to Upper case.

Ex: String str1="WELcome TO java"
String str2=str1.toUpperCase();

7. replace:-syntax: public String repalce(char old, char new);

This method replace all the appearance of old character with a new character.

Ex: String str1="JAVA"
String str2=str1.repalce('J', 'K');

This method is also replace the old string to the new string.

```
String str1="JAVA"
```

```
String str2=str1.replace('JAVA', 'KAVA');
```

8. charAt:-syntax: public String charAt(int index);

This method returns a single character located at the specified index position with a string object.

Ex: String str1="JAVA"

```
Char c=str1.charAt(2);           //output V
```

9. subString:-syntax: public String substring(int begin);

This method returns a string which is derived from the main string with the mentioned position.

This will returns the substring from the specified begin to end of the string.

Ex: String str1="welcome to java programing"

```
String str2=str1.SubString(3);//output: come to java programing
```

Syntax2: String SubString(int begin, int end);

This will returns the specified begin to the specified end

Ex: String str1="welcome t_o java programing"

```
String str2=str1.SubString(3,10);//output: come t_
```

10. trim:-syntax: public String trim();

This method is used to remove the beginning and ending of the wide space in a given string.

Ex: String str1=" JAVA PROGRAMING "

```
String str2=str1.trim();
```

Write a program to perform all the string operation

```
import java.lang.String;
```

```
import java.lang.*;
```

```
public class lab7
```

```
{
```

```
    public static void main(String args[])
```

```
    {
```

```
        String s1="java";
```

```
        String s2="PROGRAM ing";
```

```
        String s3;
```

```
        System.out.println("string1 length="+s1.length());
```

```
        System.out.println("string1 length="+s2.length());
```

```
        System.out.println("strings Concatination="+s1.concat(s2));
```

```
        if (s1.equals (s2))
```

```
        System.out.println("String are equal");
    else
        System.out.println("String are NOT equal");
    System.out.println("Strin1 UPER CASE =" + s1.toUpperCase());
    System.out.println("Strin2 LOWER CASE =" + s2.toLowerCase());
    System.out.println("Replaceing string1 " + s1 + " is=" + s1.replace('j','k'));
    System.out.println("Strings equals ignore ase=" + s1.equalsIgnoreCase(s2));
    System.out.println("Second charecter of 2nd string " + s2 + " is=" + s2.charAt(1));
    s3=s2.trim();
    System.out.println("String " + s2 + "   trimming is " + s3);
    System.out.println("String   of   " + s2 + "String   from   2nd   and   ending   from   5
is=" + s2.substring(3,6));
    }
}
```

String Buffer class

It creates string of flexible length that can be modifying in terms of both length and content. StringBuffer class object as the rights to access all the methods of string classes but the object of string class has no rights to access the methods of string buffer class.

String buffer created as:

```
StringBuffer sb=new StringBuffer("murthy")
```

Methods of string buffer class:

1.append(): This method is used to concatenating the two strings, It is affected to the current object.

```
Ex:    StringBuffer s1=new StringBuffer("vvfgc");
        StringBuffer s2=new StringBuffer("college");
        System.out.println("Append=" + s1.append(s2));
```

2.insert(): Insert the string s2 at the position 'n' of the string s1.

```
Ex:    StringBuffer s1=new StringBuffer("vvfgc");
        StringBuffer s2=new StringBuffer("fgc");
        s1.insert(3,s2);
```

3.SetLength(): This method is used to set the length from the string to 'n'.

```
Syntax: s1.SetLength(n);
Ex:    StringBuffer s1=new StringBuffer("vvfgc");
        s1.setLngth(3);
```

4.SetcharAt(): This method is used to set the nth character to the given new character.

```
Syntax: s1.SetcharAt(n,char new);
Ex:    StringBuffer s2=new StringBuffer("java");
```

```
s2.setcharAt(0,'k');//kava
```

5.reverse(): This method is used to reverse the character with in an object of the string buffer class.

Syntax: s1.reverse();

Ex: StringBuffer s1=new StringBuffer("vvfgc");
 s1.reverse();

Write a program to perform all the string buffer method

```
public class Ex
```

```
{
```

```
    public static void main(String args[])
```

```
    {
```

```
        StringBuffer s1=new StringBuffer("vvfgc");
```

```
        StringBuffer s2=new StringBuffer("college");
```

```
        System.out.println("String str1="+s1);
```

```
        System.out.println("String str2="+s2);
```

```
        System.out.println("Append="+s1.append(s2));
```

```
        System.out.println("Insertd string="+ S1.insert(3,s2));
```

```
        System.out.println("Set length of  string2="+ S2.setLength(3));
```

```
        System.out.println("Character at string s2="+ S2.SetcharAt(3,m));
```

```
        System.out.println("revers  string s2="+ s2.reverse());
```

```
    }
```

```
}
```

Vector:

“A **vector** is a class that provides the capability to implement a growable array of object”.

Vector class is constructed under the java.util.package.

Vectors are array list with extended properties which follow the dynamic and automatic addition of data at runtime.

Vector can be created as:

1. Vector v = new Vector();

This constructor creates a default vector containing no elements. The default capacity(size) is 10.

2. Vector v = new Vector(20);

This constructor creates the initial capacity of 20.

3. Vector v = new Vector(20,5);

This vector has a initial capacity of 20 and will grow in increment of 5 elements.

Methods of vector class:-

- `add()`:
 `void add(int);`
 `void add(int pos, int ele);`

This method inserts the elements into the vector.

- `capacity()`:
 `int capacity();`
It returns the current capacity of this vector.

- `remove()`:
 `remove(int pos);`
It removes the element at specified position in this vector.

- `size()`:
 `int size();`
It returns the no. of elements in this vector.

- `clear()`:
 `void clear();`
It removes all the elements from this vector.

Ex:

```
public class Ex
{
    public static void main(String args[])
    {
        Vector v = new Vector();
        for(int i=1;i<=10;i++)
            v.add(i);
        v.add(4,100);
        System.out.println("vector elements : "+v);
        v.remove(3);
    }
}
```

Wrapper classes:

“Wrapper classes are used to represent primitive values when all object is required, the wrapper class encapsulate a single value for the primitive data type”.

Before we can instantiate a wrapper class, we need to know it's name and arguments it's constructor accepts. The name of the wrapper class corresponding to each primitive data type along with the args it's constructor accepts.

Primitive data type → wrapper class → constructor args.

int → Integer → int or string

float → Float → float, double or string

double → Double → double or string

boolean → Boolean → boolean or string

byte → Byte → byte or string

char → Character → char

Wrapper classes are Boolean, Byte, Character, Short, Integer, Long, Float, Double.

Creation of wrapper class object:-

```
Boolean obj1 = new Boolean("false");
```

```
Byte obj2 = new Byte("2");
```

```
Integer obj3 = new Integer("16");
```

```
Float obj4 = new Float(12.35f);
```

```
Double obj5 = new Double("12.25");
```

We can also create the wrapper class object by using `valueOf()` method.

```
Integer n = Integer.valueOf("101011",2);           //43
```

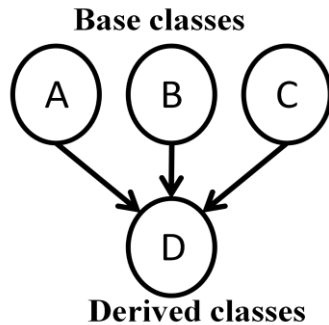
`valueOf` converts 101011 to 43 and assigns to the object 'n';

Features of wrapper classes:-

- Wrapper classes convert numeric strings into numeric values.
- The way to store primitive data in an object.
- The value of method is available in all wrapper classes except character.

Chapter-3

Interface



Java class not support multiple inheritance (two or more base class derives one derived class)

(Class D extends A, B, C is invalid)

(Class D extends, extends A, extends B, extends C is invalid)

For the above disadvantages java implements new concepts call it as interface.

“An **interface** is an alternate approach to support the concepts of multiple inheritance and contains final variable & abstract methods”.

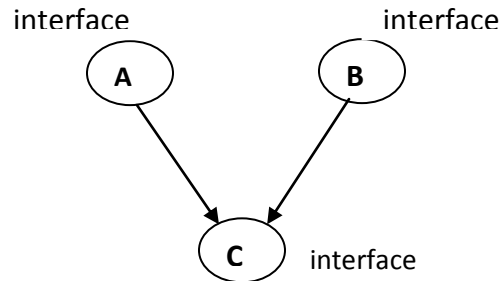
Syntax:

```
interface <interface_name>
{
    final fields
    abstract methods
}
```

All the methods in an interface are an implicitly extract and public variable declared implicitly final

Ex:

```
interface xyz
{
    int a=10;
    void add (int x, int y);
    void display ();
}
```

Extending interface

An interface can only be extended from another interface and can also extend from multiple interface.

Implementing interface:**Syntax:**

```
class <class_name> implements <interface_name1>, <interface_name2>, ....  
{  
    .....  
}
```

‘**implements**’ is a keyword used to specify that a class inherits the behaviour from an interface and as return the function definition(implements) for every method of an interface.

- A class can implements multiple interfaces.
- The interface extends interface.
- Class implements interface
- Class extends class implements interface

```
interface addition
```

```
{  
    void add();  
}
```

```
interface subtraction
```

```
{  
    void sub();  
}
```

```
interface multiplication
```

```
{  
    void mult();  
}
```

```
class Arithmetic implements addition, subtraction, multiplication
```

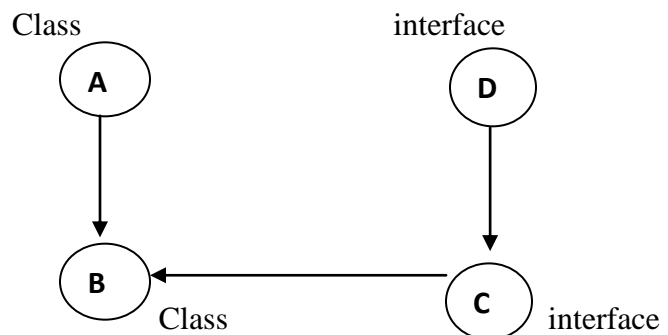
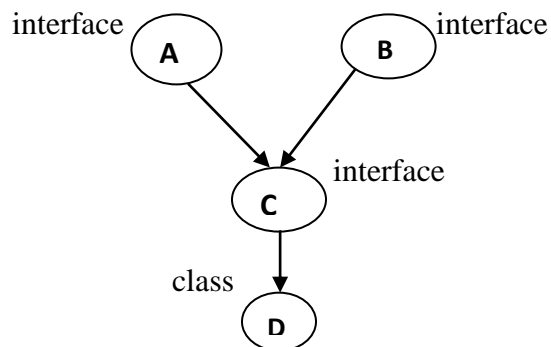
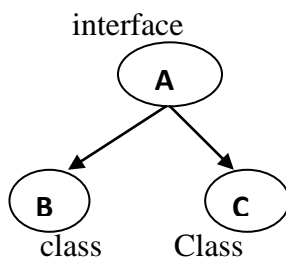
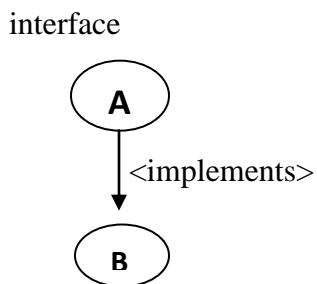
```
{  
    void add ()  
    {  
        System.out.println ("This is addition");  
    }  
    void sub ()
```

```

    {
        System.out.println ("This is subtraction");
    }
    void mult ()
    {
        System.out.println ("This is multiplication");
    }
}
public class Ex
{
    public static void main (static args[])
    {
        Arithmetic obj=new arithmetic ()
        obj.add ();
        obj.sub ();
        obj.mult ();
    }
}

```

Various form of implementing interface



Difference between class and interface**Class**

- Class contains design & Implementation of code.
- Class contain constants Variables & methods.
- Class can instantiative (Create object)
- Class declared by key Word class
- Class can extends only one Class
- Class implement interfaces

interface

- Only design
- It contains only final variable & abstract method.
- Can't instantiated (can't create object)
- Interface
- It extends more than one interface.
- It extends another interface

Difference between Abstract class & Interface**Abstract class**

- It partially implemented & partially unimplemented structure.
- It contains instance variable also.
- It should be implemented in its sub class.
- Abstract class can implement interface.
- Abstract class can extends only one class.

Interface

- Fully unimplemented structure (pure design)
- Only final variables
- All the abstract method of the interface should be implemented in its implemented class.
- An interface can't implement another interface.
- Extends more than one interface.

Access modifiers (visibility control)

There are 4 access specifies in java

- private
- public
- protected
- default (package access)

private: - It is access able only with in the class and it does not allow to access its data member and member function from outside. It is also called as most restrictive access specifies.

Ex:

```
private int x;  
private void accept ();
```

public: Any variable or methods is declared as public, than it is entire class and also visible to outside the class. It is also called as least restrictive access specifier.

Ex: public int x;
public void accept ();

protected: Any variable or method is declared as protected, it is visible only to sub class in same package and all sub class in other package.

Ex: -protected int x;
protected void accept ()

default (Package access) :

It is also called as package access. If their is no keyword specified then the variable or method is default access specifier implicitly, the default access restricted only to object in the same package and it does not allow to access from other package.

Ex: int x;
void accept();

Accessibility of access spcifiers:

Accessibility	private	Public	Protected	default
Same class	Yes	yes	yes	Yes
Same package sub class	No	yes	yes	Yes
Same package not sub class	No	Yes	No	yes
Different package sub class	No	Yes	yes	No
Different package not sub class	No	Yes	No	No

Access specifier => private, public, protected.

Access modifiers => private, public, protected, final, abstract.

Packages

“**Package**” is a collection of classes and interfaces.

Or

Package contains a set of classes and interfaces having unique names.

The advantages of packages

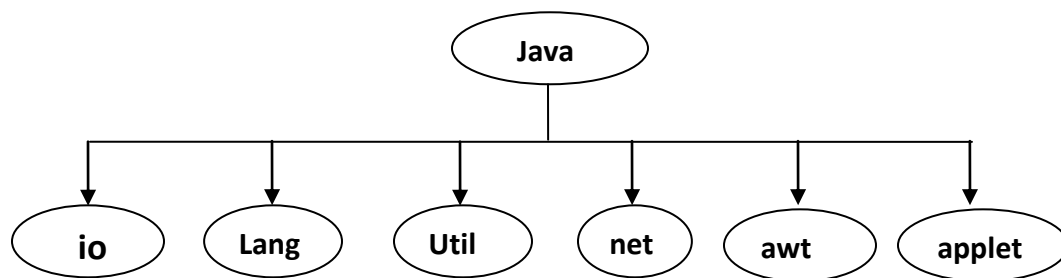
- Packages are used for code reusability
- It is the way to hide the classes
- It provides a way for separating design from coding

Types of packages:

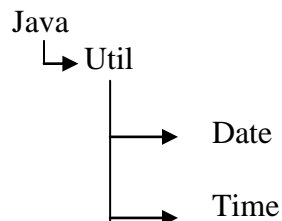
Java packages are classified into two categories:

- java API packages.
- user defined packages (built-in packages).

Java API packages



Java API packages contain the built-in classes provided by the Java. Packages are organized in a hierarchical structure. A package name should be unique for classes & interfaces.



[**Note:** - The details of all the API packages are in the 1st chapter]

User defined packages

To create a package, put the keyword `package` at the top of a Java source file. We should declare the name of the packages using the `package` keyword and followed by a package name.

Syntax: `package package_name;`

Ex: - `package pack1;`

Java uses separate directory or folder to manage the packages. We can't create a hierarchy of packages also. The package can also contain another package called sub package.

Syntax: - Package package_name1.package_name2.package_name3,.....,package_name_n;

Ex: - package pack1, pack2, pack,.....,pack n;

Steps for creating our packages

Step1: create a folder or directory which has the same name as the package name

Step2: declare the package at the beginning of the java file.

```
package pack1;
```

Step3: define the class to be put in the package & declare it as public

```
public class EX1  
{  
.....  
.....  
}
```

Step3: save the java program file in the folder as the class _name.java

```
D:\pack1>EX.java
```

Step5: compile the file using javac

```
D:\pack1>javac EX.java
```

Step6: change the present directory to previous level directory & run java file using java interprets

```
D:\>java.pack1.EX
```

Ex: - package pack1;

```
import java.io.*;  
public class Ex  
{  
    public void display ()  
    {  
        System.out.println ("this is package pack1.display fun");  
    }  
}
```

Accessing a package in another class

The keyword Import is used to access the class from the particular package.

Syntax: -

```
import package_name.*;
```

Or

```
import package_name.class_name;
```

Or

```
import pack1.pack2.pack3.pack4.*;
```

Or

```
import pack1.pack2.pack3.pack4.class_name:
```

Here pack1 is the name of the top level package,

pack2 is the name of next level package,

pack3 is the name of the next level of package. So on

The statement must terminate with a (;) semicolon

The import java.io.*; appear before class definition in program.

Ex:

```
package pack1;
```

```
import java.io.*;
```

```
public class EX
```

```
{
```

```
    public void display ()
```

```
    {
```

```
        System.out.println ("this is a pack1.package");
```

```
    }
```

```
}
```

Save the file in pack1 folder as Ex.java

Compile Ex.java file successfully then it will create EX. class file

Ex: - import pack1.EX;

```
import java.io.*;
```

```
public class xyz
```

```
{
```

```
    public static void main (String args[])
```

```
    {
```

```
        Ex obj=new Ex();
```

```
        obj.display
```

```
    }
```

```
}
```

Save this file as xyz.java in outside the pack1 folder & execute this file

```
C:\>d:
```

```
D:\>cd pack1
```

```
D:\>pack1>set path=c:\programfiles\java\jdk1.6\bin
```

```
D:\pack1>javac Ex.java
```

```
D:\pack1>cd....
D:\>javac xyz.java
D:\>java xyz
This is pack1 package
```

Program for sum of digits

```
package pack1;
public class Sumofdigits;
{
    public int sum(int n)
    {
        int rem, s=0;
        while (n!=0)
        {
            rem=n%10;
            s=s*10+rem;
            n=n/10;
        }
        return(s);
    }
}

import pack1.sum of digits;
import java.io.*;
public class Ex
{
    Public static void main (string args [])
    {
        Sum of digits.obj=new sum of digits ();
        Int ans =obj.sum (1234);
        System.out.println ("sum of digits="+ans);
    }
}
```

Hiding the class

One more advantage of package is to hide the classes from other package .if we declare the class is visible to all the packages

If we want to hide class then we can use default access

Ex:

```
package pack1;
class xyz
{
    .....
}
```

Chapter 4

Multithreaded Programming

*A multithreaded program contains two or more tasks can run simultaneously.

*Performing two or more task at same time is known as multi tasking

Two type of multi-tasking

Process based multitasking (PBM):-

Two or more program (processes) can run simultaneously is called PBM

Ex: - Running VLC, Scanning for virus, working with internet etc.....

Thread based multitasking (TBM)

Single program performing more than one task at the same time is known as thread based multitasking.

Ex: - An MS word program can check the spelling of word while we type the document.

Java program make use of both process based and thread based multitasking.

- Thread=>"A single line of execution"
- Multithreaded=>"Multiple line of execution"

Type of threads

- Daemon threads
- User threads

Daemon thread usually designed to run in the background for the purpose of servicing user thread.

User thread: - The main thread is a user thread made a variable by JVM. This thread is a launched in public static void main method. From there main thread we can create all other threads. We already work with one particular thread since from our java first program i.e. main thread

Creating threads:

Threads can created in two different ways.

- By extending the java.lang.thread.class
- By implementing the java.lang.Runnable interface

By extending threading class: -

Define a class that extends thread class and over write run (method)

class xyz extends thread

```
{
    public void run ()
    {
        System.out.println ("this is thread class extends");
    }
}
```

In Run () method will write the code that is require to be executed by new thread. The run method is the core part of thread in java. This methods is entry point and exit point for any thread we create.

import java.io.*;

import java.lang.*;

```
{
    public void run ()
    {
        For (i=1; i<=5; i++)
            System.out.println ("thread A:" +i);
        System.out.println ("exit from A");
    }
}
```

class B extends thread

```
{
    public void run ()
    {
        for (i=1; i<=5; i++)
            System.out.println ("thread B:" +i);
        System.out.println ("exit from B");
    }
}
```

public class Ex

```
{
    public static void main (string args [0])
    {
        Aobj.1=new A ();
        B obj.2=new B ();
        Obj1.start ();
        Obj2.start ();
    }
}
```

```
    }  
}
```

Output

```
Thread A: 1  
Thread B: 1  
Thread A: 2  
Thread B: 2  
Thread A: 3  
Thread B: 3  
Thread A: 5  
Thread B: 5  
Exit from B  
Exit from A
```

By implementing runnable interface:-

Define a class that implements runnable interface. The runnable interface has only one method run. This can be implemented in our class.

Syntax:-

```
interface runnable  
{  
    public void run();  
}
```

We can write the code for the run method in our class.

Example program

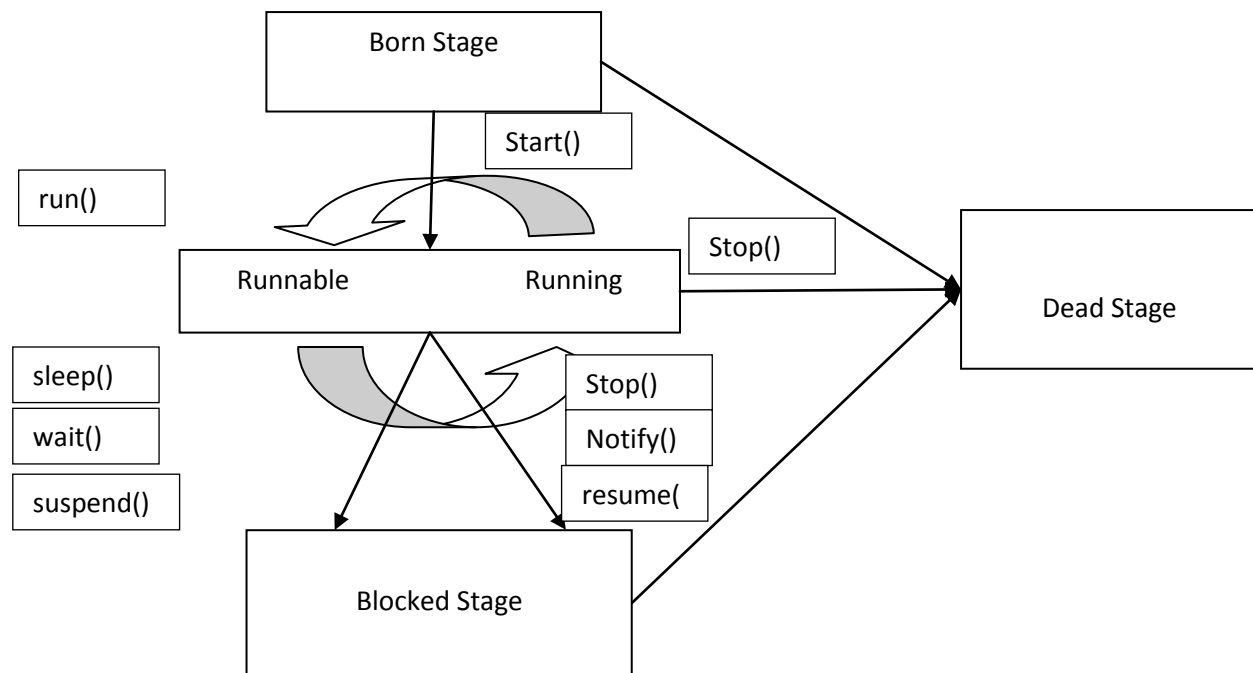
```
import java.io.*;  
import java.lang.*;  
class A implements runnable  
{  
    public void run()  
    {  
        for (int i=1;i<=5; i++)  
            System.out.println ("Thread A:" +i);  
        System.out.println ("exit from A");  
    }  
}  
class B implements runnable  
{  
    public void run()  
    {  
        for(int i=1;i<=5; i++)
```

```

        System.out.println ("Thread B:" + i);
        System.out.println ("exit from B");
    }
}
public class Ex
{
    public static void main(String args[])
    {
        A obj1 = new A();
        B obj2 = new B();
        Thread t1 = new Thread (obj1);
        Thread t2 = new Thread(obj2);
        t1.start();
        t2.start();
    }
}

```

LIFE CYCLE OF A THREAD:-



During the life time of a Thread there are many stages of a Thread can enter they are

1. Born state
2. Runnable state
3. Running state

4. Blocked state
5. Dead state

1. **Born state:-** While creating the object for a thread class and this object is not yet running. Then the Thread is new born state.
2. **Runnable state :-** When the start() is called the Thread object, the thread is in the runnable state.
3. **Running state:-** A thread currently being executed by the cpu is in the running state.
4. **Blocked state:-** When the thread is suspended, sleeping or waiting in order to satisfied certain requirement then the thread is in blocked state.
5. **Dead state:-** A running thread ends its life when it has computed its execution (Run()), then it is called a natural death.

Methods of Thread class

1. Start():- It calls the run method to being executing the Thread.
2. Stop():- To kill the Thread.
3. run():- It executes the body of the thread.
4. sleep():- It is used to move a Thread to sleep for a specified amount of a time in mile second.
5. wait():- Waiting stage of Thread until the condition.
6. Notify():- After the condition the waiting Thread automatically execute.
7. Resume():- Resumes the execution of the Thread.
8. Suspend():- It is similar to wait when using suspend the method resume will be taken
9. Yield():- It is used to stop the execution of a particular Thread
10. setName():- It sets the name of a Thread
11. getName():- It returns the name of a Thread
12. Current Thread ():- It returns the name of the currently executing the Thread.

Thread Priorities:-

Thread priority is an integer value that identifies the relative order in which it should be executed with respect to others. The Thread priority values ranging from 1 to 10 & default value is 5. But if a Thread has highest priority does not means that will execute 1st , the Thread scheduling depends on the operating system.

- **setPriority :-** It is used to set the priority that can increase the chances of the execution.
(Public final void setPriority (int value);)
t1.setPriority(10);
- **getPriority: -** This method returns the priority of a particular Thread.
Public final int getpriority ():

Ex: - int x=t1.getPriority ();

The constant priority values are:

```
public final static int  MIN_PRIORITY=1;
public final static int  NORM_PRIORITY=5;
public final static int  MAX_PRIORITY=10;
```

PROGRAM:

```
import java.io.*;
import java.lang.*;
class A extends Thread
{
    public void run ()
    {
        for(int  i=1;i<=5;i++)
            System.out.println("Thread A "+i);
        System.out.println ("Exit from A");
    }
}
class B extends Thread
{
    public void run ()
    {
        for (int  i=1;i<=5;i++)
            System.out.println ("Thread B"+i);
        System.out.println ("Exit from B");
    }
}
public class Ex
{
    public static void main (String args[])
    {
        A obj=new A ();
        B obj=new B ();
        obj2.setPriority (10);
        obj1.setPriority (1);
        obj1.setName ("1st Thread");
        obj2.setName ("2nd Thread");
        obj1.start ();
        obj2.start ();
        System.out.println ("Thread 1 Name="+obj1.getName ());
```



```
        System.out.println ("Thread 2 Name="+obj2.getName ());  
    }  
}
```

Synchronization :-

“It is the process of avoiding multiple Thread to act on same data or method”

Synchronization is used while execution time, the one Thread is under execution process there is no possibility to enter another Thread into execution.

The synchronization can be achieved in java by two ways

1. By using synchronized method
2. Synchronized block

By using synchronized method :-

The synchronized is used to synchronize the whole method

Syntax:-

```
Synchronized void withdrawal()  
{  
    .....  
    .....  
    .....  
}
```

Synchronize block :-

Synchronized block is used to synchronize block of statement in a method

Syntax:-

```
Synchronized(object)  
{  
    .....  
    .....  
    .....  
}
```

Chapter 5

EXCEPTION HANDLING

Basically errors are classified into 2 types.

1. Compile time errors.
2. Run time errors.

1. Compile Time Errors.

These errors are syntactical errors, found at the time of compilation by the compiler.

Ex: missing semicolon, missing flower bracket, missing quotes etc..

2. Run Time Errors.

Sometimes JVM would be enabling to execute statements due to some reasons. These errors are called as run-time errors.

Sometimes programs may produce wrong results due to wrong logic or may terminate due to some reasons.

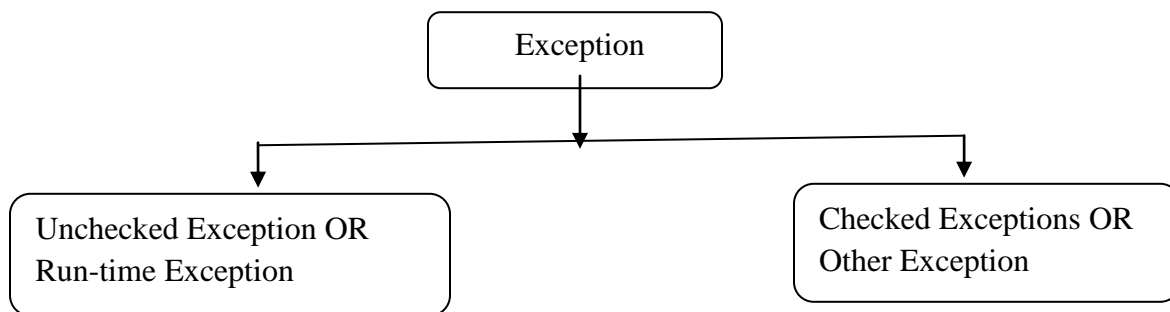
- Divide by zero (1/0).
- Accessing an element out of bounds of an array.
- Reading data from an already closed file.

Exception: “An error occurs at runtime is called an Exception.”

Exception Handling

“The mechanism of handling runtime errors or exceptions is called Exception handling.”

Types of Exceptions



Unchecked Exceptions (run-time exceptions)

The compiler doesn't handle the exceptions at compile- time, those exceptions are called unchecked exceptions. These are checked by the JVM.

Example for Unchecked Exception Classes:

- ArithmeticException
- ArrayIndexOutOfBoundsException
- ClassCastException

Checked Exception(Other Exceptions)

The compiler handles at the time of compilation, those exceptions are called Checked Exceptions.

Example for Checked Exception Classes:

- IOException
- FileNotFoundException
- illegalAccessException

Exception Handling performs the following Techniques

1. Hit the Exception :- Finding the problem.
2. Throw the Exception :- Finding that error as occurred.
3. Catch the Exception :- receiving the error information.
4. Handle the Exception :- Taking Actions.

Exception Handling Keywords

try
catch
throw
throws
finally

try :

```
try
{
    .....
    .....    //java statements
}
```

The try block consists of a executable statements that can possible to throw exceptions implicitly.

try block throw the exceptions implicitly, if the exception occurs in this block.

catch block:

```
try
{
    .....
    .....    //java statements
}
catch(exception_type e)
{
    .....
    .....    //processing of exception
}
```

catch block catches the exception thrown by the try block.

- A catch block consists of the keyword catch followed by a single parameter that identify the type of exception and 'e' is the object for the class Exception.

- Using this object (e) we can print details of Exception.
- Exception type can be the type of exception.
- “Exception class is the super class for all other Exception class types.”
- e.getMessage(): It prints the detailed information about the Exception that as occurred.

Note: If we are using catch block, we must use try block.

Example:

```
import java.io.*;
public class Ex
{
    public static void main(String args[])
    {
        int d=0;
        try
        {
            System.out.println(10/d);
        }
        catch(ArithmeticException e)
        {
            System.out.println(e.getMessage());
        }
    }
}
```

Multiple catch blocks.

Syntax: try

```
{
    .....
    .....
}
catch(Exception_type1 e)
{
    .....
    .....
}
catch(Exception_type2 e)
{
    .....
    .....
}
catch(Exception_type3 e)
{
    .....
    .....
}
```

We can also use multiple catch blocks to handle different types of Exceptions. If the try block throw an Exception, that will matches the catch block Exception type and processing the Exception in the particular catch block.

```
Ex: try
{
    int y=a[1]/a[0];
    System.out.println("Y="+y);
    a[2]=30;
}
catch(ArithmeticException e)
{
    System.out.println("Don't divide by zero");
}
catch(ArrayIndexOutOfBoundsException e)
{
    System.out.println("It is Array Index Out of Bounds");
}
catch(ArrayStoreException e)
{
    System.out.println(e.getMessage());
}
}
```

[Note: Checked Exception classes and Unchecked Exception classes see the Text book or browse in the Internet.]

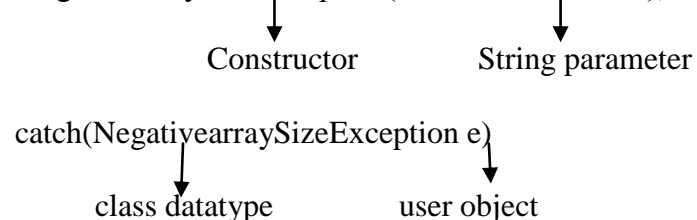
throw keyword (clause)

The “throw” keyword is used when the programmer wants to throw an exception explicitly and handle it using catch block.

“throw keyword is used to throw an exception explicitly.”

Syntax: throw new Throwable Exception Object;

Ex: throw new NegativeArraySizeException("Plz Enter the size");



Example:

```
import java.io.*;
public class Ex
{
    public static void main(string args[])
    {
        int size;
        int[] a=new int[10];
        size=Integer.parseInt(args[0]);
        try
        {
            if(size<0)
                throw new negativeArraySizeException("Plz enter positive size");
            for(int i=0;i<size;i++)
                a[i]=i+1;
        }
        catch(NegativeArraySizeException e)
        {
            System.out.println(e);
        }
    }
}
```

throws keyword(clause)

“The throws keyword is used when the programmer doesn’t wants to handle the exception in the method(inside the method) and throw it out of method.”

The method must declare it using the ‘throws’ keyword. ‘throws’ keyword appears at the end of a method signature ,if multiple exceptions are there ,we can separate it with comma operator(.).

Syntax:

```
return_typemethod_name(parameter_list)throws Exception_type1,Exception_type2,.....
{
```

```
    .....
    .....
}
```

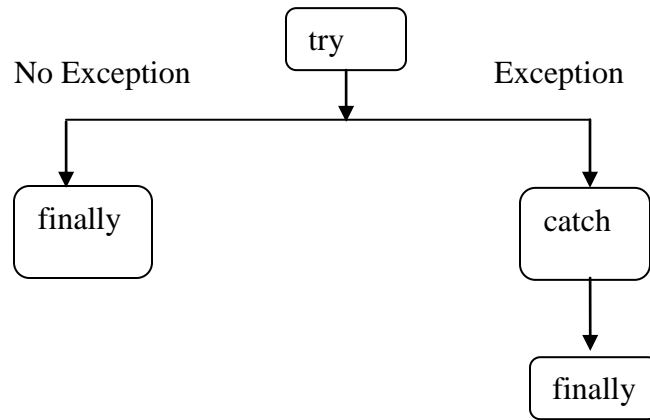
Ex:

```
public class Ex
{
    public static void main(String args[])throws IOException,NumberFormatException
    {
        .....
        .....
    }
}
```

finally block

“A finally block of code always executes whether an Exception as occurred or not.”

Finally block appears at the end of the catch block or at the end of try block.

**Syntax:**

```
try
{
    .....
    .....
}
catch(Exception_type e)
{
    .....
    .....
}
finally
{
    .....
    ..... //finally block always executes.
}
```

finally block is commonly used for:

- Closing a file.
- Closing a result set.
- Closing the connection established with connection.

[Note: finally block is optional.]

Ex:

```
import java.io.*;
public class Ex
{
    public static void main(String args[])
    {
        int size;
        int[] a=new int[10];
        size=Integer.parseInt(args[0]);
        try
        {
            if(size<0)
                throw new NegativeArraySizeException("Plz enter positive size");
            for(int i=0;i<size;i++)
                a[i]=i+1;
        }
        finally
        {
            System.out.println("This is Finally Block");
        }
    }
}
```

Creating Our Own exception Classes

When we want to create an Exception class of our own, simply make your class as sub class Exception.

Ex:

```
import java.io.*;
class Vvfgc extends NegativeArraySizeException
{
    Vvfgc()
    {
        super("plz enter +ve array size");
    }
}
public class Ex
{
    public static void main(String args[])
    {
        int size;
        int[] a=new int[10];
        size=Integer.parseInt(args[0]);
        try
        {
            if(size<0)
                throw new Vvfgc();
        }
    }
}
```



```
        System.out.println("This is +ve Array Size");
    }
    catch(Vvfgc e)
    {
        System.out.println(e);
    }
}
}
```

Here we have created our own exception class called as Vvfgc, which is a sub class of NegativeArraySizeException. By running the program with negative value, the output will be, Vvfgc: plz enter +ve array size.

Features of Exception Handling in Java.

- Separate error-handling code from normal code.

In traditional programming error detection, reporting and handling often makes the code more confusing, but Java provides an easier solution to the problem of error management i.e Exceptions.

- Propagation of Errors.

Another feature of Java Exception is the ability to propagate error reporting of the call of methods using exception classes.

Note:

- * We can have nesting of try statements, which means, we can have try-catch block within try block.
- * If we keep System.exit(0); in try block, then it will not go to the finally block.

Chapter 6

Applets

“Applets are small java program developed for internet applications”.

HTML page are communicate with the user on internet, it can use a special java program called an Applet to communicate and respond to the user.

Applet=Java byte code +HTML

Definition: “Applet is a small application that is embedded into an HTML page, which is accessed and transported over the Internet.”

Applets can be used to provide dynamic user-interface and a variety of graphical effects for web pages.

Differences between Applets and applications

Applets	Applications
❖ Applets do not have main() method	❖ Applications have main() method
❖ Applets can be embedded into HTML pages	❖ Applications cannot be embedded into HTML pages
❖ Applets only are executed within a Java browser or applet viewer	❖ Application program can be compatible executed using Java interpreter

Life cycle of an applet

To create an applet, we have applet class of Java.applet package

Applets have four lifecycle methods:

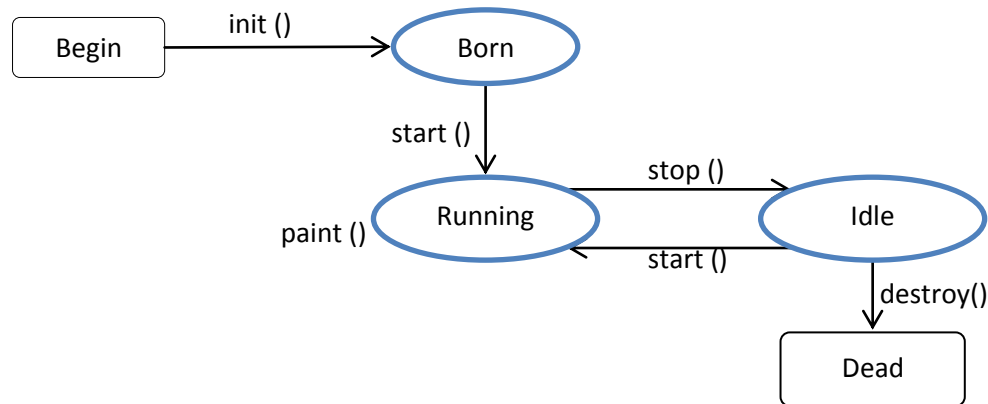
init(): It's the method used to initialize the applet before it gets loaded.

start():It's used to provide startup behavior for an applet.

stop():Its used to stop any operations which are started using start() method.

destroy(): Its used when an applet has finished its execution.

These methods are called automatically by the JVM .In the Applets life cycle methods, start and stop methods can be called multiple times whereas init() and destroy() can be called only once in the entire life span of an Applet.



init(): This is the first method called by an applet once it has been loaded by the browser. The applet is born at this stage and loading images, initial values, setup colors etc.

```
public void init()
{
    ..... //action
}
```

start(): This method occurs automatically after the applet is initialized, any calculations and processing of the data should be done in this method

```
public void start()
{
    ..... //action
}
```

stop(): stop() method occurs automatically when we leave the page. if user minimized the webpage, the this method is called and when user again comes back to the page the start() method is called.

```
public void stop()
{
    ..... //action
}
```

destroy(): This method is called when an applet is being terminated from memory. The stop() method will always be called before destroy().

```
public void destroy()
{
    ..... //action
}
```

paint(): This method is not a part of an applet life cycle. Applets moves to the display state when it has to perform some output operations on the screen. It occurs immediately after applet runs into start() method.

```
public void paint(Graphics g)
{
    ..... //User interface elements and display strings
}
```

E.g. An sample applet template

```
import java.applet.*;
import java.awt.Graphics;
public class Ex extends Applet
{
    public void init()
    {
        ..... //called first time applet is executed
    }
    public void start()
    {
        ..... //called after init()
    }
    public void stop()
    {
        ..... //called when webpage disappears
    }
    public void destroy()
    {
        ..... //called when applet is being removed from memory.
    }
    public void paint(Graphics g)
    {
        g.drawString("welcome",100,200);
        ..... //other graphical functions
    }
}
```

Writing and executing a simple Applet

```
import java.applet.Applet;
import java.awt.graphics;
public class Ex extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("Hello Applet",50,60);
    }
}
```

The two import statements.

Every applet that we create must be a subclass of Applet

The graphics class is a part of Abstract Window Toolkit package

drawString() is a method of the Graphics class

This method is used to print the given msg at x, y location on the applet

void drawString(String msg, int x, int y);

compile: Save the above file as Ex.java and compile using 'javac'.

After successful compilation 'Ex.class' byte code file is created.

Creating an HTML page for embedding an Applet

```
<html>
<body>
<applet      code="Ex.class"  width=440  height=200>
</applet>
</body></html>
```

Run this .html page using either appletviewer through command prompt

- ❖ ...\\bin>appletviewer <filename.html>
- Or
- ❖ Open the <filename.html> via web browser.

<Applet > Tag

```
<Applet [code=Applet classfile]  [codebase=codebase URL]
      [Alt=Alternative Text]      width=pixels    height=pixels
      [Align=Alignment]          [Name=Applet Name]>
</Applet>
```

Code: Indicates the name of the class file.

Codebase: Indicates the pathname where the class file are contained.

Alt: if any problem with running applets an alternate text is displayed.

Width and Height: Used to mention the size of an applet window in pixels.

Align: The applet will be aligned on the page.

Left, Right, Top, TextTop, Middle, AbsMiddle, Bottom

Name: Used to specify a name for the applet.

Passing Parameters to Applets

To setup and handle parameters in an applet. We need two things

- A special parameter tag in the HTML file.
- Code in our applet program to read those parameters

In HTML file that contains the embedded applet

<param>tag is used to indicate parameters.

This tag has two attributes **Name&Value**

```
<PARAM Name=font      Value="Arial">
<PARAM Name=size      Value="36">
```

In our applet program these parameters are passed when it is load init() method of applet

```
String name;
String size;
public void init()
{
    Name=getParameter("font");
    Size=getParameter("size");
}
e.g:
import java.applet.Applet;
import java.awt.Graphics;
public class Ex extends Applet
{
    int x,y;
    public void init()
    {
        x=Integer.parseInt(getParameter("xvalue"));
        y=Integer.parseInt(getParameter("yvalue"));
    }
    public void paint(Graphics g)
    {
        g.drawString("Vidyavahini",x,y);
    }
}
<html>
    <body>
        <applet code="Ex.class" width=400 height=400>
            <PARAM      Name=xvalue      value=100>
            <PARAM      Name=yvalue      value=100>
        </applet>
    </body>
</html>
```

Aligning the Applet Display

The Align attribute defines how the applet will be aligned on the page. An applet can be embedded inside an HTML that can also contain some text to be displayed with applet

```
<html>
    <body>
        <applet code="Ex.class"      width=100  height=100 Align=left>
            <PARAM      Name=xvalue      value=100>
            <PARAM      Name=yvalue      value=100>
        </applet>
        Vidyavahini First Grade College<br>
        Bachelor of Computer Applications<br>
```

```
SIT Extension<br>
Tumkur
</body>
</html>
```

Displaying Numerical values in applet

To display numeric values there is no method to display directly. We should convert the numeric values into string and use drawstring method to display.

String valuesOf() method is used to convert numeric to string

```
int num=100;
String s=String.valueOf(num);
g.drawString(s,20,20);
```

Getting Input from User

The user interface components like textbox, button, checkbox etc. for developing interactive application.

We have used another part of the java swing library for entering data, display messages or both using JOptionPane class.

To read the input data showInputDialog() method is used. The entered data will be always string. if we want numeric values then we should convert it.

```
import java.awt.*;
import java.applet.Applet;
import java.swing.*;
public class Ex extends JApplet
{
    String str1,str2, sumstr;
    int num1,num2,sum;
    public void init()
    {
        str=JOptionPane.showInputDialog("Enter a Number");
        num1=Integer.parseInt(str1);
        num2=Integer.parseInt(str2);
        sum=num1+num2;
        sumstr=String.valueOf(sum);
    }
    public void paint(Graphics g)
    {
        g.drawString(sumstr,50,50);
    }
}
```

update() Method

This method is used to redraw the portion of its window. Update() method performs all necessary display activities.

```
public void update(Graphics g)
{

}

public void paint(Graphics g)
{
    update(g);
}
```

repaint() method:

When every our applet needs to update the information displayed in its window, it simply calls repaint() is used to repaint the window.

void repaint(int left, int width, int height);

Specified region that will be repainted.

void repaint();

Causes window to be repainted.

Advantages of Applets

Applets can run on windows, MacOS and Linux platform.

Applets can work all the version of the JavaPlugin.

Applets can work without security approval.

Applets are supported by most web browsers.

Applets are quick to load when returning to web pages.

Disadvantages of Applets

Java plugin is required to run applet.

Java applet requires JVM, so first time it takes some small startup time.

It is difficult to design and build good user interface in applets compared to other technologies.

Awt Controls

- ✓ Labels
- ✓ Buttons
- ✓ Textfields
- ✓ Check Boxes
- ✓ Lists

All these controls are subclasses of component class. To add controls to a window we can use add() methods.

Labels

```
Label ()  
Label (String str)  
Label (String str,int align)  
Label L1=new Label ("Student Name:");  
add (L1);
```

Text fields

```
TextField()  
TextField(int <no of char>)  
TextField(String str)  
TextField(String str, int <no of char>)  
TextField t=new TextField();  
add(t);
```

Buttons

```
Button()  
Button(String Str);  
Button b=new Button("submit");  
add(b);
```

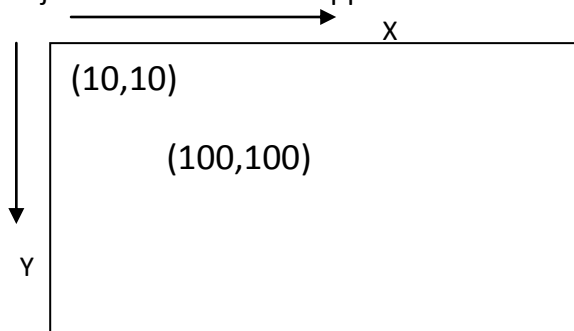
Graphics Programming

The graphics class with in the abstract windowing tool kit (AWT)makes it possible for a java programmer to draw simple geometric shapes, print text and position images with in the borders of a component, such as a applet, frame, panel or canvas.

The graphics class used draw lines, shapes, characters and images to the screen inside an applet.

The graphics co-ordinate system

To draw an object on the screen, you call one of the drawing methods available in the graphics class. All the drawing methods have arguments representing end points, sorting locations of the objects as values in the applet's co-ordinate system.



The drawing methods of the graphics class

DrawRect()	-Draws a rectangle
DrawRoundRect()	-Draws a rounded-corner rectangle
DrawLine()	-Draws a line
drawOval()	-Draws an oval
drawArc()	-Draws an arc
draw3Drect()	-Draws a 3D rectangle
drawPolygon()	-Draws a polygon connecting point to point
drawPolyline()	-Draws a line connecting point to point
fill3DRect()	-Draws a filled 3D rectangle
fillArc()	-Draws a filled arc
fillOval()	-Draws a filled oval
fillPolygon()	-Draws a filled polygon
fillRect()	-Draws a filled rectangle
fillRoundRect()	- Draws a filled rectangle with corners

Drawing Lines

```
void drawLine(int x1, int y1, int x2, int y1);
```

The drawLine() takes four arguments (x1, y1) co-ordinates of the starting point and (x2, y2) co-ordinates of the ending point.

```
public void point(Graphics g)
{
    g.drawLine(50,50,200,200);
}
```

Drawing Rectangles

There are 3 kinds of rectangles :

- Plain rectangles.
- Rounded rectangles.
- Three dimensional rectangles.

```
public void point(Graphics g)
{
    g.drawRect(100,100,200,200);
    g.fillRect(200,200,300,300);
}

public void point(Graphics g)
{
    g.drawRoundRect(20,20,60,60,50,50);
    g.fillRoundRect(120,20,60,60,25,25);
}

public void point(Graphics g)
{
    g.draw3DRect(20,20,60,60,true);
    g.draw3DRect(120,20,60,60,false);
}

void drawRect(int x1,int y1,int x2,int y2);
```

```
void drawRoundRect(int x1,int y1,int x2,int y2,int
xarc,int yarc);
void draw3DRect(int x1,int y1,int x2,int y2,boolean);
}
```

Drawing circles and ellipses

```
void drawoval(int x1,int y1,int x2,int y2);
public void point(Graphics g)
{
    g.drawOval(20,20,60,60);
    g.fillOval(120,20,100,60);
}
```

Drawing Arcs

An arc is a part of an Oval, the easiest way to think of an arc is as a complete Oval.

```
void drawArc(int x1,int y1,int x2,int y2,int rectangle, int
arcangle);
public void point(Graphics g)
{
    g.drawArc(50,50,80,60,45,120);
    g.fillArc(150,50,80,60,45,120);
}
```

Drawing polygons

Polygons are shapes with an unlimited number of sides. To draw a polygon, we need a set of 'x' and 'y' co-ordinates.

The drawPolygon() and fillPolygon() methods takes 3 arguments:

- An array of integers representing x-co-ordinates.
- An array of integers representing y-co-ordinates.
- An integer for the total number of points.

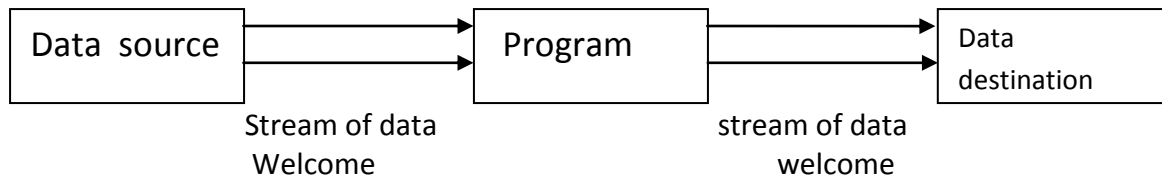
```
public void paint (Graphics g)
{
    int x1[]={125,125,150,150,200,200};
    int y1[]={200,100,100,175,175,200};
    g.drawPolygon(x1, y1, x1.length);
}
```

Chapter 7

I/O stream

Stream: “stream is a flow of information from a source to a destination”

Ex:- a) A keyboard to a monitor.
b) A monitor to a file on a hard disk.
c) A file on a hard disk to a monitor.



Streams are mainly used to move data from one place to another.

The java.io package contains a large No of stream classes that provide capabilities processing all type of data.

Streams are basically divided into 2 categories:

- Byte stream classes
- Character stream classes

✓ **Byte stream classes**

To read and write 8-bytes, programs should use the byte streams. The byte streams are defined by using two classes hierarchies.

- Input stream class
- Output stream class

✓ **Character stream classes**

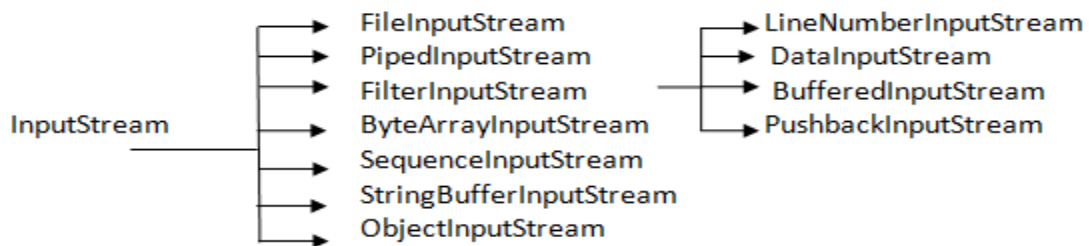
To read and write 16-bytes, programs should use the character streams. The character streams are defined by using two classes hierarchies.

- Reader classes
- Writer classes

➤ **Byte stream classes**

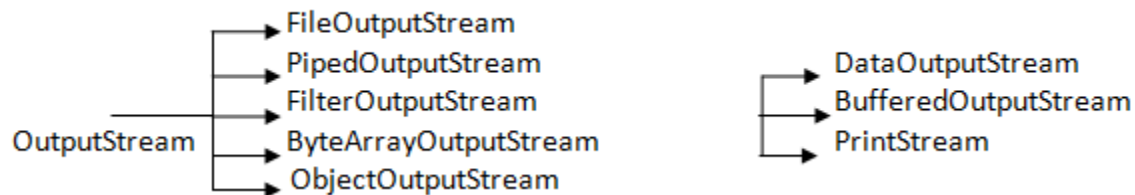
Input stream

Input stream is an abstract class that provides the framework from which all other input stream are derived.



OutputStream

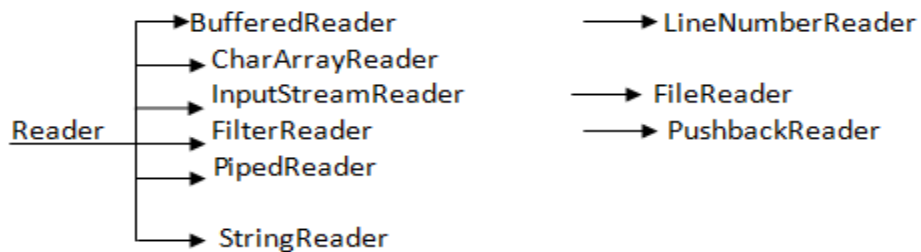
OutputStream is an abstract class that provides the framework from all output stream is derived.



Character Stream Classes

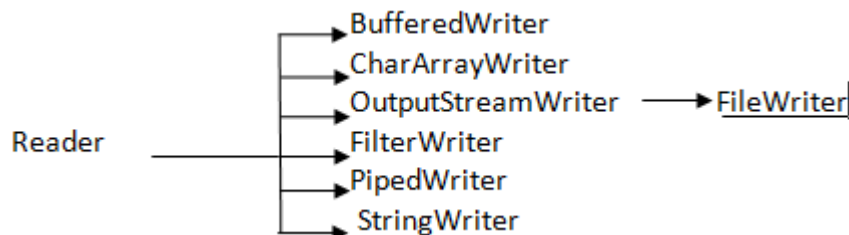
Reader

Reader is an abstract class that provides the framework from which all other reader streams are derived.



Writer

Writer is an abstract class that provides that framework from which all other writer stream are



derived.

Creating Files

Whenever we need to store data permanently then we use the concept of files. We can create a file, write data into file, read data from an existing file. We can copy the contents of our file to another file. To perform all these operation we use i/o stream.

⇒ **Prog. To illustrates the use of FileInputStream**

```
import java.io.*;
class Ex
{
    public static void main(String args[])throws IOException
    {
        try
        {
            FileInputStream fis=new FileInputStream("ster.txt");
            int c;
            While((c=fis.read())!=-1)
            System.out.print(c);
            fis.close();
        }
        Catch(FileNotFoundException e)
        {
            System.out.println("File Not Found");
        }
    }
}
```

FileInputStream

⇒ **A prog. to read data from the user & writer this into a file.**

```
import java.io.*;
public class Ex
{
    public static void main(String args[])
    try
    {
        DataInputStream ds=new DataInputStream(System.in);
        System.out.println("Enter the data");
        String s=ds.readLine();
        FileInputStream f=new FileInputStream("xyz.txt");
        byte b[]=s.getBytes()
        f.write(b);
        f.close();
    }
    Catch(IOException e)
    {
        System.out.println("Exception Occurred");
    }
}
```

```
    }  
}
```

⇒ **A Prog. to read the data from the file & Print on screen.**

```
    public class Ex  
{  
    public static void main(String args[])  
  
    {  
    try  
    {  
    FileInputStream f=new FileInputStream("xyz.txt");  
    int size=f.available();  
    byte b[]=new byte[size];  
    f.read(b);  
    String s=new String(b);  
    System.out.println("the file context are:"+s);  
    f.close();  
    }  
    Catch(IOException e)  
    {  
    System.out.print("Exception Occurred");  
    }  
    }  
}
```