Chapter 1: Fundamentals of JAVA

1.1 History of JAVA

1) Java originated at Sun Microsystems, Inc. in 1991.
2) It was conceived by James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan at Sun Microsystems, Inc.
3) It was developed to provide a platform-independent programming language.
4) Sun was founded on February 24, 1982
5) **Sun Microsystems, Inc.** was a company that sold computers, computer components, computer software and information technology services
6) The Sun name is derived from the initials of the Stanford University Network
7) On January 27, 2010, Sun was acquired by Oracle Corporation for US$7.4 billion
8) Java was designed by Sun Microsystems in the early 1990s to solve the problem of connecting many household machines together. This project failed because no one wanted to use it.
9) Then it was redesigned to work with cable TV. This project also failed because the cable companies decided to choose a competing system
10) When the World Wide Web became popular in 1994, Sun realized that Java was the perfect programming language for the Web.
11) Early in 1996 (late 1995?) they released Java (previously named Oak) and it was an instant success! It was a success, not because of marketing, but because there was a great need for a language with its characteristics.

1.2 Java is every where

1) JAVA resides in mobiles, client machines, server machines, embedded devices, smart phones.
2) It shares the same basic features of the language and libraries.
3) Principle of Java: Write Once, Run Anywhere(WORA)

1.3 Versions of JAVA

Version	Release Date	No of Classes	No of Packages
JAVA1.0	May-1996	212	8
JAVA1.1	Feb-1997	503	23
JAVA2.0	Dec-1998	1520	59
JAVA5.0	Sep-2004	3562	166
JAVA6.0	Dec-2006	3792	203
JAVA7.0	Jul-2011	3977	209

1.4 Features of JAVA

1) **Simple**
   - Looks familiar to existing programmers: related to C and C++
   - Omits many rarely used, poorly understood, confusing features of C++, like operator overloading, multiple inheritance etc
   - Has no header files and eliminated C pre-processor
   - Eliminates much redundancy (e.g. no structs, unions)
   - No pointers
2) **Object Oriented Language**
   - Java is an object-oriented language, which means that you focus on the *data* in your application and *methods* that manipulate that data, rather than thinking strictly in terms of procedures
   - Java comes with an extensive set of classes, arranged in *packages* that you can use in your programs.
   - Java supports encapsulation, polymorphism and Inheritance.
3) **Distributed**

- Java is Distributed Language Means because the program of java is compiled onto one machine can be easily transferred to machine and Executes them on another machine because facility of Bytes Codes So java is Specially designed For Internet Users which uses the Remote Computers For Executing their Programs on local machine after transferring the Programs from Remote Computers or either from the internet.
- Java provides an extensive library of classes for communicating, using TCP/IP protocols such as HTTP and FTP. This makes creating network connections much easier than in C/C++. You can read and write objects on the remote sites via URL with the same ease that programmers are used to when read and write data from and to a file. This helps the programmers at remote locations to work together on the same project.

4) Interpreted

- The Java compiler generates *byte-codes*, rather than native machine code.
- To actually run a Java program, you use the Java interpreter to execute the compiled byte-codes.
- Java byte-codes provide an architecture-neutral object file format.
- The code is designed to transport programs efficiently to multiple platforms

5) Robust

- Java has been designed for writing highly reliable or robust software
- language restrictions (e.g. no pointer arithmetic and real arrays) to make it impossible for applications to smash memory (e.g overwriting memory and corrupting data)
- Java does automatic garbage collection, which prevents memory leaks
- extensive compile-time checking so bugs can be found early

6) Secure

- Security is an important concern, since Java is meant to be used in networked environments.
- Java's memory allocation model is one of its main defences against malicious code (e.g can't cast integers to pointers)

7) Portable

- compiler generates bytecodes, which have nothing to do with a particular computer architecture
- easy to interpret on any machine
- standard libraries hide system differences

8) Multi-threaded

- Java allows multiple concurrent threads of execution to be active at once
- This means that you could be listening to an audio clip while scrolling the page and in the background downloading an image

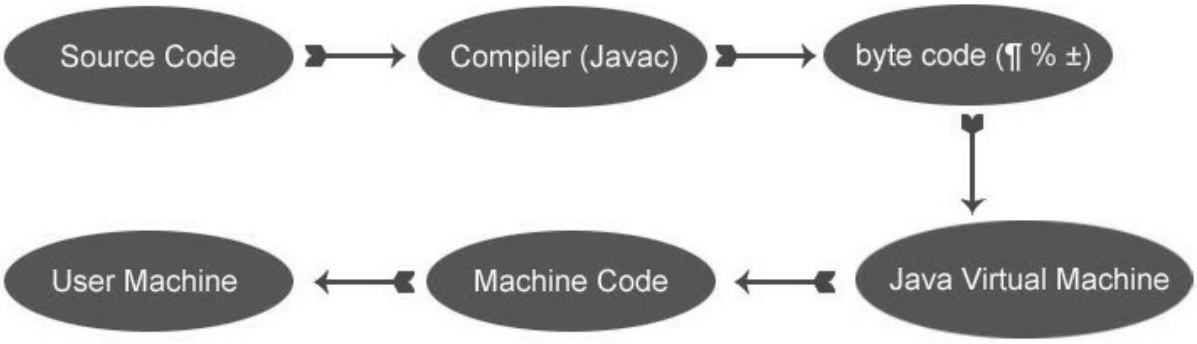
9) Garbage Collector

- It frees memory allocated to objects that are not being used by the program any more

- In Java the GC runs automatically, but you can also call it explicitly with System.gc() and *try to* force a major garbage collection

1.5 Compile and Run

- Unlike many other programming languages including C and C++ when Java is compiled, it is not compiled into platform *specific*, rather into platform independent byte code.
- When the source code (.java files) is compiled, it is translated into byte codes and then placed into (.class) files.
- This byte code is distributed over the web and interpreted by Java virtual Machine (JVM) on whichever platform it is being run.
- The JVM is the environment in which Java programs execute.
- A JVM can either interpret the bytecode one instruction at a time or the bytecode can be compiled further for the real microprocessor using what is called a just-in-time compiler.
- JIT compilers represent a hybrid approach, with translation occurring continuously, as with interpreters, but with caching of translated code to minimize performance degradation



1.6 Installation and Setup

- **Step 1: goto http://www.oracle.com/technetwork/java/javase/downloads/index.html**

- Step 2: Click on the JAVA image as shown in the above picture.

- Step 3: Accept the license

x x86	106.64 MB	jdk-7u13-linux-i586.rpm
x x86	92.97 MB	jdk-7u13-linux-i586.tar.gz
x x64	104.77 MB	jdk-7u13-linux-x64.rpm
x x64	91.69 MB	jdk-7u13-linux-x64.tar.gz
OS X x64	143.71 MB	jdk-7u13-macosx-x64.dmg
ris x86 (SVR4 package)	135.55 MB	jdk-7u13-solaris-i586.tar.Z
ris x86	91.95 MB	jdk-7u13-solaris-i586.tar.g
ris x64 (SVR4 package)	22.54 MB	jdk-7u13-solaris-x64.tar.Z
ris x64	14.96 MB	jdk-7u13-solaris-x64.tar.gz
ris SPARC (SVR4 package)	135.83 MB	jdk-7u13-solaris-sparc.tar
ris SPARC	95.28 MB	jdk-7u13-solaris-sparc.tar
ris SPARC 64-bit (SVR4 package)	22.89 MB	jdk-7u13-solaris-sparcv9.t
ris SPARC 64-bit	17.58 MB	jdk-7u13-solaris-sparcv9.t
dows x86	88.74 MB	jdk-7u13-windows-i586.ex

- Step 4: Choose your version
- Step 5: Now you have downloaded **jdk-7u45 –windows-x64.exe,** click on this file to run the setup

1. 7 Verifying Installation

- Go to the specified path, where you install your JAVA.
- You will see a JAVA folder
- Inside JAVA folder you will see two folders, one is for jdk and other is for jre
- **JDK:**
  - **Java Developer Kit** contains tools needed to develop the Java programs
  - These tools could be Compiler (javac.exe), Application Launcher (java.exe), etc
- **JRE:**
  - **Java Runtime Environment**
  - It contains **JVM** (**J**ava **V**irtual **M**achine) and Java Package Classes(Java Library)
- **JVM**
  - JVM is platform dependent
  - The ***Java Virtual Machine*** provides a platform-independent way of executing code
  - Java Virtual Machine interprets the byte code into the machine code depending upon the underlying operating system and hardware combination.

1.8 Facts related to JAVA

1 Java is a case sensitive language like C and C++
2 Java is nearly 100% object oriented language
3 In java, it is not possible to make a function which is not a member of any class (as we can do in C++)

1.9 First Program

Following is the first program in JAVA. You can type the code in any text editor (like notepad). Since the language is case sensitive please notice the uppercase and lowercase letters:

```
public class HelloWorld
{
  public static void main(String [] args)
  {
     System.out.println("Hello World");
  }
} // do not put semicolon here
```

1.10 Explanation of first program

- Unlike C++, java classes can be public.
- Java classes can be public, private, protected or default. Only inner classes can be private or protected (see Later).
- The keyword class specifies that we are defining a class.
- The name of a class is spelled exactly as the name of the file (Case Sensitive).
- All java programs begin execution with the method named main().
- Declaring main method as public means that it is accessible from outside the class so that the JVM can find it when it looks for the program to start it.
- It is necessary that the method is declared with return type void (i.e. no arguments are returned from the method).
- The main method contains a String argument array that can contain the command line arguments.
- String args[], which is actually an array of java.lang.String type, and it's name is args here. It's not necessary to name it args always, you can name it strArray or whatever you like.
- When we run a Java program from command prompt, we can pass some input to our Java program. Those inputs are stored in this String args array.
- The brackets { and } mark the beginning and ending of the class
- The program contains a line 'System.out.println("Hello World");' that tells the computer to print out on one line of text namely 'Hello World'.

- The semi-colon ';' ends the line of code.
- The double slashes '//' are used for comments that can be used to describe what a source code is doing
- System is a built-in class present in java.lang package
- This class has a final modifier, which means that, it cannot be inherited by other classes
- It contains pre-defined methods and fields, which provides facilities like standard input, output, etc.
- out is a static final field (ie, variable)in System class which is of the type PrintStream (a built-in class, contains methods to print the different data values).
- out here denotes the reference variable of the type PrintStream class
- println() is a public method in PrintStream class to print the data values
- Hence to access a method in PrintStream class, we use out.println() (as non static methods and fields can only be accessed by using the reference variable)

1.11 Compile and Run your Java Program

The JDK contains documentation, examples, installation instructions, class libraries and packages, and tools. Following are the steps to create, compile and run your java program:

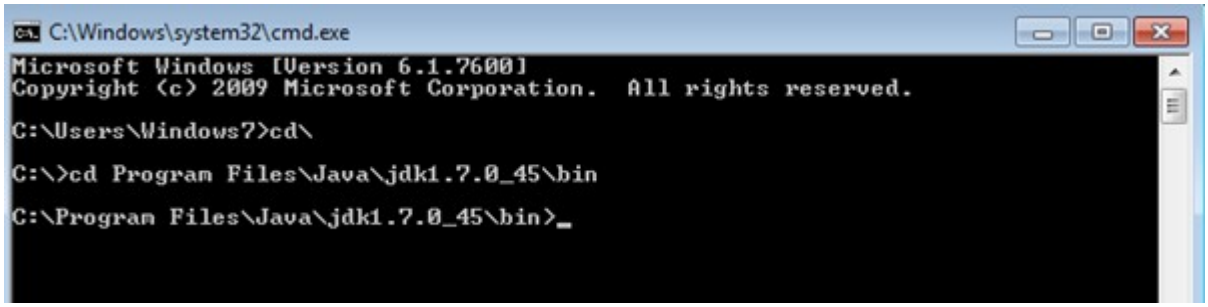
1 You must save your source code with a .java extension.
2 The name of the file must be the name of the class contained in the file.
3 Save the program With .java Extension
4 Open MSDOS, reach to Java Bin folder
5 Compile the file by typing javac <filename with .java extension>.
6 Successful Compilation, results in creation of .class containing byte code
7 Execute the file by typing java <filename without extension>

Example:

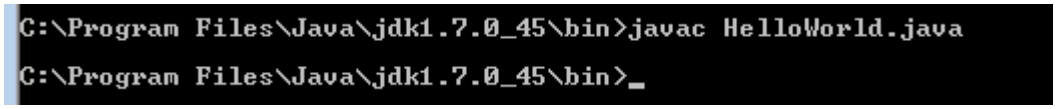
Step 1: Open Notepad Editor with Run as Administrator and type the above program.

Step 2: Save the program as "HelloWorld.java" including double quotes at C:\Program Files\Java\jdk1.7.0_45\bin or specify the path to bin folder where you installed Java.\

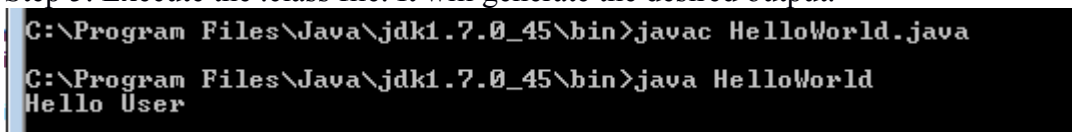
Step 3: Open command prompt with Run as Administrator and reach to bin folder.



Step 4: Compile the program. After successful compilation HelloWorld.class will generate.



Step 5: Execute the .class file. It will generate the desired output.



Check Yourself:

Objective Type Question:

1. Which of the following is correct about main method in java?

a) Must be declared as public.

b) It must not return any value.

c) Must be declared as static.

d) Must accept string as a parameter.

e) Above all are compulsory.

f) All are optional.

2. Which of the following is correct?

(a)

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

```
        System.out.println(“Hello World!”);
    }
}
```

(b)

```
class MainClass {
    public static void main(String[] args) {
        System.out.println(“Hello World!”);
    }
}
```

(c)

```
 class MainClass {
     static public int main(String[] args) {
          System.out.println(“Hello World!”);
          return 1;
    }
}
```

(d) Both option (a) and (b)

1. Why we use array as a parameter of main method

a) it is syntax

b) Can store multiple values

c) Both of above

d) None of above

4. The Java Program is enclosed in a class definition.

a) True

b) False

5. What is the range of data type short in Java?

a) -128 to 127

b) -32768 to 32767

c) -2147483648 to 2147483647

d) None of the mentioned

6. What is the range of data type byte in Java?

a) -128 to 127

b) -32768 to 32767

c) -2147483648 to 2147483647

d) None of the mentioned

7. Which of the following are legal lines of Java code?

1. int w = (int)888.8;

2. byte x = (byte)100L;

3. long y = (byte)100;

4. byte z = (byte)100L;

a) 1 and 2

b) 2 and 3

c) 3 and 4

d) All statements are correct.

8. Which of these is necessary condition for automatic type conversion in Java?

a) The destination type is smaller than source type.

b) The destination type is larger than source type.

c) The destination type can be larger or smaller than source type.

d) None of the mentioned

9. If an expression contains double, int, float, long, then whole expression will promoted into which of these data types?

a) long

b) int

c) double

d) float

10. What is the error in this code?

byte b = 50;

b = b * 50;

a) b can not contain value 100, limited by its range.

b) * operator has converted b * 50 into int, which can not be converted to byte without casting.

c) b can not contain value 50.

d) No error in this code

Subjective Type Question:

1. WAP to print below output:

Hello

World

2. WAP to print below output:

Hello 100

3. What are local variables?

4. What are instance variables?

5. What is the default value of byte datatype in Java?

6. What is the default value of float and double datatype in Java?

7. When a byte datatype is used?

8. What are primitive types in java?

9. What happens if you dont initialize an instance variable of any of the primitive types in Java?

10. What will be the output of the following statement?

System.out.println (“1” + 3);

Chapter 2: Start Learning Java

2.1 Comments in JAVA

1. *Block style* comments begin with /* and terminate with */ that spans multiple lines.
2. *Line style* comments begin with // and terminate at the end of the line.
3. *Documentation style* comments begin with /** and terminate with */ that spans multiple lines. They are generally created using the automatic documentation generation tool, such as javadoc

2.2 Identifiers

- Source code entities such as classes and methods need to be named so that they can be referenced from elsewhere in the code.
- An identifier consist of a letter, digits, underscore and dollar sign $
- The name cannot start with digit
- Almost any valid identifier can be chosen to name a class, method or other source code entity.
- Some identifiers are reserved for special purposes; they are known as reserved words.
- Reserved words are also known as keywords
- Following is the list of keywords:

abstract	default	goto	package	this
assert	do	if	private	throw
boolean	double	implements	protected	throws

break	enum	import	public	transient
byte	else	instanceof	return	true
case	extends	int	short	try
catch	false	interface	static	void
char	final	long	strictfp	volatile
class	finally	native	super	while
const	float	new	switch	
continue	for	null	synchronized	

2.3 Data types

- A type identifies a set of values (and their representation in memory) and a set of operations that transform these values into other values of that set.
- Java is strongly typed language
- Java classifies types as primitive types, user-defined types and array types
- A primitive type is a type that is defined by the language and whose values are not objects

Primitive Type	Size
boolean	**implementation dependent**
char	**16 bits (stores unicode)**
byte	**8 bits**

short	**16 bits**
int	**32 bits**
long	**64 bits**
float	**32 bits**
double	**64 bits**

- A user-defined type is a type that is defined by the developer using a class , an interface, or an enum.
- User defined types are also known as reference types because a variable of that type stores a reference to a region of memory that stores an object of that type.
- In contrast, variables of primitive types store the values directly; they don't store references these values.

- An array type is a special reference type that signifies an array, a region of memory that stores values in equal size and contiguous slots, which are commonly referred to as elements.

2.4 Variables

4) Programs manipulate values that are stored in memory, which is symbolically represented in source code through the use of the variables feature.
5) A variable is a named memory location that stores some type of value.
6) Variables that store references are often referred to as reference variables.
7) Variables must be declared before they are used.
8) Variables are not explicitly initialized to any values.
9) Variables are either initialized to default values or remain uninitialized (depending upon whether they are declared in the class or within methods).

2.5 Constants

1. boolean constant consist of reserved word *true* or reserved word *false*.

2. character constant consist of a single Unicode character surrounded by a pair of single quotes

3. integer constant consists of a sequence of digits

4. If the constant is to represent a long integer value, it must be suffixed with an uppercase L or lowercase l.
5. If there is no suffix the constant represents a 32 bit integer (an int).
6. Integer constant can be specified in the decimal, hexadecimal, octal or binary format
    a. 127
    b. 0x7f
    c. 0177
    d. 0b01101100

2.6 Conversion Rules

- Byte integer to short integer, integer, long integer, floating point or double
- Short integer to integer, long integer, floating point, or double
- Character to integer, long integer, floating point, or double
- Integer to long integer, floating point or double precision floating point
- Long integer to floating point or double precision floating point
- Floating point to double

2.7 Conversion via cast operator

- Byte integer to character
- Short integer to byte integer or character
- Character to byte integer or short integer
- Integer to byte integer, short integer or character
- Long integer to byte, short, character or integer
- Floating point to byte integer, short, character, integer, or long
- Double to byte, short, character, integer, long, or float

Check Yourself:

Objective Type Question:

1. What is the stored in the object obj in following lines of code?

box obj;

a) Memory address of allocated memory of object.

b) NULL

c) Any arbitrary pointer

d) Garbage

2. Which of these keywords is used to make a class?

a) class

b) struct

c) int

d) None of the mentioned

3. Which of the following is a valid declaration of an object of class Box?

a) Box obj = new Box();

b) Box obj = new Box;

c) obj = new Box();

d) new Box obj;

4. Which of these operators is used to allocate memory for an object?

a) malloc

b) alloc

c) new

d) give

5. What is the output of this program?

```
class main_class {
    public static void main(String args[])
    {
        int x = 9;
        if (x == 9) {
```

```
            int x = 8;
            System.out.println(x);
        }
    }
}
```

a) 9

b) 8

c) Compilation error

d) Runtime error

6. Classes are useful because they

a) permit data to be hidden from other classes

b) can closely model objects in the real world

c) brings together all aspects of an entity in one place

d) all of the above.

7. The .dot operator connects the following two entities :

a) a class member and a class object

b) a class object and a class

c) a class and a member of that class

d) a class object and a member of that class

8. What is the output of this program?

```
class box {
    int width;
    int height;
    int length;
```

```
}
class mainclass {
    public static void main(String args[])
    {
        box obj1 = new box();
        box obj2 = new box();
        obj1.height = 1;
        obj1.length = 2;
        obj1.width = 1;
        obj2 = obj1;
        System.out.println(obj2.height);
    } }
```

a) 1

b) 2

c) Runtime error

d) Garbage value

9.. Which of these statement is incorrect?

a) Every class must contain a main() method.

b) Applets do not require a main() method at all.

c) There can be only one main() method in a program.

d) main() method must be made public.

10. Which method can be defined only once in a program?

a) main() method.

b) finalize method

c) static method.

d) private method

Subjective Type Question:

1. WAP to define a class to represent a Bank Account. Following are the members.

Data members:

a)Name of the depositor

b)Account number

c)Type of account

d)Balance amount in the account

Member Functions

a)To assign initial values

b)To deposit an amount

c)To withdraw an amount after checking the balance

d)To display name and balance

2. WAP to create a class called Box having three data members length, width and height. Create another class with main method. Create object of Box class and initialize length, width and height and print it.

3. What kind of variables a class can consist of?

4. List the three steps for creating an Object for a class and explain them with example?

5. Explain the difference between instance variable and a class variable. Provide example to explain them.

Prepared By: Ankit Desai

Chapter 3: Classes and Objects

3.1 Structured vs Object Oriented Programming
3.2 Defining classes
3.3 Creating Objects
3.4 Instance member variables
3.5 Instance Member Functions

3.1 Structured vs Object Oriented Programming

4. Structured programs separate data from behaviors. This separation makes it difficult to model real world entities and often leads to maintenance headaches when programs become complex.

5. In object oriented programming, classes and objects combine data and behaviors into program entities. Programs based on classes and objects are easier to understand and maintain.

3.2 Defining Classes

- A class is a template for manufacturing objects.
- Classes generalize real world entities
- Class is a means to achieve encapsulation. Encapsulation is an act of combining properties and methods related to an object.
- Class is a blue print of an object.
- Class is a means to create data type (user defined data type)
  Example:

```
class Box{
        int length;
        int breadth;
        int height;
        public void setDimension(int l,int b, int h)
        {
                length=l;
                breadth=b;
                height=h;
        }
        public void showDimension()
        {
                System.out.println("Length="+length);
                System.out.println("Breadth="+breadth);
                System.out.println("Height="+height);
```

```
        }
}
```

In the above example, Box is a class with three member variables length, breadth, height and two member functions setDimension(), showDimension().

3.3 Creating Objects

- An instance of a class is known as object
- Object is a real world entity, as it can map real world entity by storing its properties in member variables.
- Instance means object

Box b1;

- In the above statement b1 is not an object. b1 is object reference. b1 can be used to represent an object of Box class.

Box b1=new Box();

- The new operator allocates memory to store the object whose type is specified by new's operand, which happens to be Box() in our example.
- The parentheses that follow Box signify a constructor.
- When a constructor ends, new returns a reference to the object so that it can accessed elsewhere in the program.
- Reference variable referring to an object can also be termed as 'handle' to the referring object
- Object consumes memory
- The process of creating objects from a class is called instantiation.

3.4 Instance Member Variables

- Instance variables and instance methods, which belong to objects, are collectively called instance members.
- Instance member variables are variables which belong to an instance (or object).
- Instance member variables are declared in the class
- Their life time depends on the life time of object.
- Every instance has a separate set of instance member variables
- Instance member variables
- The dot '.' notation with an object reference is used to access Instance Members.
- Instance members can also be initialized during declaration

- Instance member variables are initialized by default with its default value.

3.5 Instance Member Functions

- Instance member functions are functions that are defined inside the class to provide methods to access instance members of an object
- They can be used via object only.
- State of an object should be accessed or altered via instance member functions only (OOP promoted concept)

Check Yourself:

Objective Type Question:

1.What is the output of the following program?

```
class MyClass
{
   static int j=func1();
   static int i=10;
   static int func1()
   {
      return i;
}

public static void main(String[] args)
   {
      System.out.println("i="+i);
      System.out.println("j="+j);
   }
}
```

A.

i=10

j=0

B.

i=10

j=10

C. Compilation Error

D. Run Time Error

2. Which of these is a wrapper for data type int?

a) Integer

b) Long

c) Byte

d) Both a & b

3. Which of the following methods is a method of wrapper Integer for obtaining hash code for the invoking object?

a) int hash()

b) int hashcode()

c) int hashCode()

d) Integer hashcode()

4. Which is true about an anonymous inner class?

a) It can extend exactly one class and implement exactly one interface.

b) It can extend exactly one class and can implement multiple interfaces.

c) It can extend exactly one class or implement exactly one interface.

d) It can implement multiple interfaces regardless of whether it also extends a class.

5. Which statement is true about a static nested class?

a) You must have a reference to an instance of the enclosing class in order to instantiate it.

b) It does not have access to nonstatic members of the enclosing class.

c) It's variables and methods must be static.

d) It must extend the enclosing class.

6. Which of the following statements are true?

a) The Integer class has a String and an int constructor

b) The Integer has a floatValue() method

c) The wrapper classes are contained in the java.lang.Math package

d) The Double class has constructors for type double and float

7. What will happen when you attempt to compile and run the following code?

```
public class WrapMat{
   public static void main(String argv[]){
   Integer iw = new Integer(2);
   Integer iw2 = new Integer(2);
   System.out.println(iw * iw2);
   System.out.println(iw.floatValue());
  }
}
```

a)Compile time error

b) Compilation and output of 4 followed by 2.0

c) Compilation and output of 4 followed by 2

d) Compile time error, the Integer class has no floatValue method

8. What will happen when you attempt to compile nad run the following code?

```
public class TwoEms {

   public static void main(String argv[]){

      Object[] oa = new Object[3];

      oa[0] = new Integer(1);
```

```
        int i = oa[0];
        System.out.print(i);
       }
    }
```

a) Compile time error an array cannot contain object references

b) Compile time error elements in an array cannot be anonymous

c) Compilation and output of 1

d) Compile time error Integer cannot be assigned to int

e) Compilation and output of the memory address of the Integer instance

9. Which of these is a super class of wrappers Long, Character & Integer?

a) Long

b) Digits

c) Float

d) Number

10. Which of the following is valid code?

a) System.out.println(Integer.toBinaryString(4));

b) System.out.println(Integer.toOctalString(4));

c) System.out.println(Integer.add(2,2));

d) Float[] ar = new Float[] { new Float(1.0), new Float(2.1)};

Subjective Type Question:

1. WAP to create a class called Employee having empid and salary as data members. Create an objects of this class and count the no: of employee working in an organization using static keyword.

2. Can you access non static variable in static context?

3. What are the Data Types supported by Java? What is Autoboxing and Unboxing?

4. Where we can declare static variable?

5. Can we declare a local variable with in static block, like below?

```
public class TestVariable {
        static{
         int i=9;
         System.out.println("print  "+i);
    }
```

6. Can we use an instance variable with in static block ?

```
public class TestVariable {
    int i=9;
        static{
            System.out.println("print  "+i);
    }
```

7. What are the different type of inner classes?

8. Why do we need wrapper class?

Chapter 4

Static Members and Wrapper Classes

4.1 Static Members

1) Static members in java are those members which are qualified with the keyword static.
2) We can have static member variables, static member functions, and static inner classes but not static local variable in any method.
3) Static members are also known as class members (and not instance members)

4) Static members are those that belong to a class as a whole and not to a particular instance (object).
5) A static variable will get memory as soon as class is loaded.
6) Static members in the class can be accessed either by using the class name or by using the object reference, but on the other hand, instance members can only be accessed via object references.

4. 2 Static Member Variables

- Member variables of a class can be static (use keyword static during declaration of variable).
- Static variables are by default initialized to its default value (for int it is zero, for float it is 0.0, for object reference it is null, etc)
- Static variable has a single copy for the whole class and does not depend on the objects
- Since static members can exist without any object, they can also be accessed with the class name and dot(.) operator.
- Static member variable is used to hold value of a property that belongs to the whole class and not to a specific object. For example, in Account class accountNo, balance, etc are instance member variables and belongs to every instance of the class but rateOfInterest should be static member variable as it belongs to whole class.

Example:

```
class Account{
        private int accountNo;
        private float balance;
        private static float rateOfInterest;
        public void setAccountNo(int a)
        {
                accountNumber=a;
        }
        public float calculateInterest(int time)
        {
                return(balanace*rateOfInterest*time/100.0);
        }
}
```

4.3 Static Member Functions

- Static functions defined inside the class are qualified with the keyword static
- Static function can only access static members of the same class
- Static function can be invoked using class name and dot operator

- Static member functions are specially needed when static member variables of class are not accessible from outside the class (may be private).
- The main() method in Java is also static member method.

Modify the previous example

```
class Account{
        private int accountNo;
        private float balance;
        private static float rateOfInterest;
        public void setAccountNo(int a)
        {
                accountNumber=a;
        }
        public float calculateInterest(int time)
        {
                return(balanace*rateOfInterest*time/100.0);
        }
        public static float getRateOfInterest()
        {
                return(rateOfInterest);
        }
}
```

4.4 Static inner classes

- Java allows to define a class inside another class known as inner class, further this class can be static. (Outer class cannot be static)
- Static inner classes can be used with the help of name of outer class and dot(.) operator.

4.5 Wrapper Classes

Java is an object-oriented language and as said everything in java is an object, but what about primitive types. They are sort of left out in the world of Objects, that is, they cannot participate in the object activities. As a solution to this problem, Java allows you to include the primitives in the family of objects by using what are called **wrapper classes**.

- There is a wrapper class for every primitive data type in Java.
- Wrapper class encapsulates a single value for the primitive data type
- For instance the wrapper class for int is Integer, for float is Float, and so on
- Following are the primitive types and their corresponding Wrapper class in java:

boolean Boolean

byte	→	Byte
char	→	Char
short	→	Short
int	→	Integer
long	→	Long
float	→	Float
double	→	Double

4.6 Methods of Wrapper Classes

valueOf()

It is a static method and returns object reference of relative wrapper class.

Example:

Float f1=Float.valueOf("3.14f");

valueOf() method takes a string argument, converted it into its equivalent primitive type and assigned to an object of corresponding wrapper class, then returns the reference of this object.

Example

Integer i2=Integer.valueOf("101011",2);

In this example, valueOf() method takes two arguments, first argument is a string and second argument is its base

parseXxx()

parseXxx() is static method of wrapper classes where Xxx is any primitive type(like parseInt(), parseDouble() etc). It returns the corresponding primitive type.

Example

int i = Integer.parseInt("123");

double d= Double.parseDouble("3.14");

Check Yourself:

Objective Type Question:

1) Which option is correct:

```
public void foo( boolean a, boolean b)
{
if( a )
   {
      System.out.println("A"); /* Line 5 */
   }
else if(a && b) /* Line 7 */
   {
      System.out.println( "A && B");
   }
else /* Line 11 */
   {
      if ( !b )
      {
         System.out.println( "notB") ;
      } else
      {
         System.out.println( "ELSE" ) ;
      }
   }
}
```

a) If a is true and b is true then the output is "A && B"

b) If a is true and b is false then the output is "notB"

c) If a is false and b is true then the output is “ELSE”

d) If a is false and b is false then the output is “ELSE”

2) switch(x)

```
{
   default:
      System.out.println(“Hello”);
}
```

Which two are acceptable types for x?

1. byte
2. long
3. char
4. float
5. Short
6. Long

a) 1 and 3

b) 2 and 4

c) 3 and 5

d) 4 and 6

3) public void test(int x)

```
{
   int odd = 1;
   if(odd) /* Line 4 */
   {
      System.out.println(“odd”);
```

```
    }
    else
    {
        System.out.println("even");
}
}
```

Which statement is true?

a) Compilation fails.

b) "odd" will always be output.

c) "even" will always be output.

d) "odd" will be output for odd values of x, and "even" for even values.

4)

```
public class While
{
    public void loop()
    {
        int x= 0;
        while ( 1 ) /* Line 6 */
        {
            System.out.print("x plus one is " + (x + 1)); /* Line 8 */
        }
}
    }
}
```

Which statement is true?

a) There is a syntax error on line 1.

b) There are syntax errors on lines 1 and 6.

c) There are syntax errors on lines 1, 6, and 8.

d) There is a syntax error on line 6.

5) public class Loop {

```
   public static void main(String[] args) {
        Integer a=012,b;
        for(b=0;b<=a;b++);
        System.out.print(b);
   }
}
```

What will be the output of above java program?

a)10

b)11

c)12

d) Compiler error

6) public class Loop {

```
   public static void main(String[] args){
        int b=0;
        do{
          int a=2;
          b++;
System.out.println(a++);
        }
```

```
        while(b!=3);
    }
}
```

What will be the output of above java program?

a)2
  2
  2

b)3
  3
  3

c)3
  4
  5

d) Declaration is not allowed here, Compiler error

7)
```
public class Loop {
    public static void main(String[] args){
        for(int a=0,b=0;a+b<5;a++,++b){
            System.out.print(a^b);
        }
    }
}
```

What will be the output of above java program?

a)111

b)000

c)1111

d)Compiler error

8) public class Loop {

```
    public static void main(String[] args){
        int i=0;
        for(;i<4;i++){
          System.out.println(i<2);
        }
    }
}
```

What will be output of above program?

a)Infinite loop

b) true

false

false

false

c) true

true

false

false

d)Compiler error

9)public class Loop {

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

```
    int i=0;
        for(i++;i<4;i++){
          System.out.println(~i);
        }
       }
       }
```

What will be the output of above java program?

a)-1

-3

b)-2

c)-2

-3

-4

d)Compiler error

10)public class Loop {

```
    public static void main(String[] args){
        int a[]={06,07,010,011};
        for(int value:a)
            System.out.println(value);
    }
}
```

What will be the output of above java program?

a)6

7

10

11

b)6

7

8

9

c)06

07

010

011

d)Compiler error

Subjective Type Question:

1) Write a program which shows how to generate factorial of a given number using recursive function.

2) Write a program which check the number is even or odd. Number is given as command line argument.

3) Write a Program that checks number is zero, positive or negative using nested if statement. Number is given as command line argument

4) Write a program which assigns a grade based on the value of a test score: an A for a score of 90% or above, a B for a score of 80% or above, and so on.

5) Write a program which declares an int named month whose value represents a month. The code displays the name of the month, based on the value of month, using the switch statement.

6) Write a program that calculates the number of days in a particular month using switch statement.

7) Write a program which displays the number of the month based on the value of the String named month.

8) Write a program which accept three numbers from the user and check which number is greatest using nested if-else statement.

9) Write a program which accepts number from the user and prints sum of its digits.

10) Write a program which accepts number from the user and prints sum of its first and last digit only.

Chapter 5:

Command Line Arguments

And

Control Instruction

5.1 What are command line arguments?

A Java application can accept any number of arguments from the command line. When a java program is launched with the help of application launcher java.exe through command prompt we can supply arguments to the program. Program can use this information for any further processing.

5.2 How to Supply Command Line arguments?

Let us suppose we have a java program Hello.class, and we want to supply names during launch of the program. Following is the command:

java Hello Rahul Aditya Roshan

The above line is a command to run java byte code Hello.class and three arguments Rahul, Aditya and Roshan are passed as data/information.

5.3 String array in main is use to receive command line arguments

We have an array of type String in the main method as formal arguments. This array is used to receive command line arguments. Considering the above command, Strings Rahul, Aditya and Roshan are stored as an array values represented by args[0], args[1] and args[2], where args is the name of reference to an array of String.

Example

```
public class Hello{
        public static void main(String[] args) {
                for (int i=0;i<args.length;i++)
                        System.out.println("Hello, "+args[i]);
  }
}
```

Output of the above program

Hello, Rahul

Hello, Aditya

Hello, Roshan

In Java all array provides a length property whose value is the length of an array.

5.4 Arguments are String

Whatever you supplied through the command line, it gets received by String array, so arguments are always Strings. Sometimes it is needful to convert String information into other type.

Example: Program to add N numbers. Numbers are supplied through command line

```
public class AddNumbers {

public static void main(String[] args){

        int sum=0;

        for(int i=0;i<args.length;i++)

                sum=sum+Integer.parseInt(args[i]);

        System.out.println("Sum is "+sum);

  }

}
```

5.5 What are Control Instructions?

A program needs to be able to repeat code and skip over sections of the code. High-level languages like Java provide constructs such as for, if, and while for *structured programming*. Control structures like for, if, and while make it easy to see where control of a program goes, because programs can enter each loop only at the beginning and leave at the end.

5.6 Classification of Control Instructions

1. Decision making statements (if, if-else, switch, ?:)
2. Statement for loop blocks (while, do-while, for and enhanced for)

5.7 Syntax of if

```
if(condition) {
        <statement-1>
        <statement-2>
        <statement-3>
        <statement-...>
}

if(condition) {
        <statement-1>
}
```

Or you can omit the curly brackets when there is only one statement in the if block.

```
if(condition)

        <statement-1>
```

Example

```
class Example {

 public static void main(String []args) {

        int x=5;

        if(x>0)

         System.out.println(x+" is positive");

 }
```

```
}
```

What would be the output of the following code?

```
class Example {
 public static void main(String []args) {
        int x=5;
        if(x)
         System.out.println(x+" is positive");
 }
}
```

It will generate an error as condition in the if statement must be of boolean type.

5.8 Syntax of if else

```
if(condition)
 <statement-1>
else
 <statement-1>
```

When there are multiple statements, curly brackets are necessary to form a block:

```
if(condition){
 <statement-1>
 <statement-2>
 <statement-3>
}
else {
 <statement-1>
 <statement-2>
```

```
}
```

Example

```
class Example {
 public static void main(String []args) {
        int x=5,y=6,min;
        if(x<y)
                min=x;
        else
                min=y;
 }
}
```

5.9 Syntax of Conditional Operator

Condition **?** then_statement **:** else_statement;

Example

```
class Example {
 public static void main(String []args) {
        int x=5,y=6,min;
        min=x<y ? x : y;
 }
}
```

5.10 Syntax of switch

The *switch* conditional structure implements a conditional structure with multiple branches.

It replaces a more efficient structure than multiple *if-then-else* clauses.

Following is the syntax of switch

```
switch(variable){
 case constant_value1:
        <statements>
        break;
case constant_value2:
        <statements>
        break;
...
default:
        <statements>
}
```

- switch, case, default and break are keywords

Example

```
String s1=null;
int x=2;
switch(x){
 case 1:
        s1="case 1";
        break;
case 2:
        s1="case 2";
        break;
default:
        s1="default";
}
```

5.11 Loop

Loops are used to iterate code in any number of times. Java supports four varieties of loop:

- while
- do while
- for
- enhanced for

<u>while</u>

This is an entry control loop

```
while(condition){
 <statement 1>
 <statement 2>
```

<statement ...>

}

do while

This is an exit control loop

do{

<statement 1>

<statement 2>

<statement ...>

} while(condition);

for

This is an entry control loop

for(initialization; condition; iteration){

<statement 1>

<statement 2>

<statement ...>

}

Enhanced for

for(variable; iterable collection){

<statement 1>

<statement 2>

<statement ...>

}

Example

public static void main(String[] args)

{

```
        int []num={1,2,3,4};
        for(int x : num)
            System.out.println(x);
    }
```

5.12 Controlling loops

- Loops can be controlled optionally via keywords break and continue.
- The *break* statement stops the current loop block implemented by *for, do-while*, *while and enhanced for*.
- Also, if it is used in a *switch* structure, it will close the last *case* clause.
- The *continue* instruction will suspend the current iteration of a loop block, *for, enhanced for, do-while*, *while-do* and will execute the next iteration.

Check Yourself:

Objective Type Question:

1) Which option is correct:

```
public void foo( boolean a, boolean b)
{
if( a )
   {
      System.out.println("A"); /* Line 5 */
   }
else if(a && b) /* Line 7 */
   {
      System.out.println( "A && B");
   }
else /* Line 11 */
   {
```

```
    if ( !b )
    {
       System.out.println( "notB") ;
    } else
    {
       System.out.println( "ELSE" ) ;
    }
  }
}
```

a) If a is true and b is true then the output is "A && B"

b) If a is true and b is false then the output is "notB"

c) If a is false and b is true then the output is "ELSE"

d) If a is false and b is false then the output is "ELSE"

---

2) switch(x)

```
{
  default:
    System.out.println("Hello");
}
```

Which two are acceptable types for x?

1. byte
2. long
3. char
4. float
5. Short
6. Long

a) 1 and 3

b) 2 and 4

c) 3 and 5

d) 4 and 6

3) public void test(int x)

{

int odd = 1;

if(odd) /* Line 4 */

{

System.out.println(“odd”);

}

else

{

System.out.println(“even”);

}

}

Which statement is true?

a) Compilation fails.

b) “odd” will always be output.

c) “even” will always be output.

d) “odd” will be output for odd values of x, and “even” for even values.

4) public class While

{

```
   public void loop()
   {
      int x= 0;
      while ( 1 ) /* Line 6 */
      {
         System.out.print("x plus one is " + (x + 1)); /* Line 8 */
      }
}
}
}
```

Which statement is true?

a) There is a syntax error on line 1.

b) There are syntax errors on lines 1 and 6.

c) There are syntax errors on lines 1, 6, and 8.

d) There is a syntax error on line 6.

---

5)
```
public class Loop {
   public static void main(String[] args) {
      Integer a=012,b;
      for(b=0;b<=a;b++);
      System.out.print(b);
   }
}
```

What will be the output of above java program?

a)10

b)11

c)12

d) Compiler error

6) public class Loop {

```
  public static void main(String[] args){
      int b=0;
      do{
        int a=2;
        b++;
System.out.println(a++);
      }
      while(b!=3);
  }
}
```

What will be the output of above java program?

a)2

2

2

b)3

3

3

c)3

4

5

d) Declaration is not allowed here, Compiler error

7) public class Loop {

```
   public static void main(String[] args){
       for(int a=0,b=0;a+b<5;a++,++b){
          System.out.print(a^b);
       }
   }
}
```

What will be the output of above java program?

a)111

b)000

c)1111

d)Compiler error

```
8) public class Loop {
   public static void main(String[] args){
      int i=0;
      for(;i<4;i++){
        System.out.println(i<2);
      }
   }
}
```

What will be output of above program?

a)Infinite loop

b) true

   false

   false

false

c) true

true

false

false

d)Compiler error

9)public class Loop {

public static void main(String[] args){

int i=0;

for(i++;i<4;i++){

System.out.println(~i);

}

}

}

What will be the output of above java program?

a)-1

-3

b)-2

c)-2

-3

-4

d)Compiler error

10)public class Loop {

public static void main(String[] args){

```
        int a[]={06,07,010,011};
        for(int value:a)
            System.out.println(value);
    }
}
```

What will be the output of above java program?

a)6
  7
  10
  11

b)6
  7
  8
  9

c)06
  07
  010
  011

d)Compiler error

Subjective Type Question:

1) Write a program which shows how to generate factorial of a given number using recursive function.

2) Write a program which check the number is even or odd. Number is given as command line argument.

3) Write a Program that checks number is zero, positive or negative using nested if statement. Number is given as command line argument

4) Write a program which assigns a grade based on the value of a test score: an A for a score of 90% or above, a B for a score of 80% or above, and so on.

5) Write a program which declares an int named month whose value represents a month. The code displays the name of the month, based on the value of month, using the switch statement.

6) Write a program that calculates the number of days in a particular month using switch statement.

7) Write a program which displays the number of the month based on the value of the String named month.

8) Write a program which accept three numbers from the user and check which number is greatest using nested if-else statement.

9) Write a program which accepts number from the user and prints sum of its digits.

10) Write a program which accepts number from the user and prints sum of its first and last digit only.

Chapter 6: Packages, Access Modifiers And Constructors

6.1 Java Packages

- Packages are nothing more than the way we organize files into different directories according to their functionality, usability as well as category they should belong to.
- Files in one directory (or package) would have different functionality from those of another directory.
- For example: files in java.io package do something related to I/O, but files in java.net package give us the way to deal with the Network
- Packaging also help us to avoid class name collision when we use the same class name as that of others
- The benefits of using package reflect the ease of maintenance, organization, and increase collaboration among developers

6.2 How to create Package?

- Suppose we have a file called HelloWorld.java, and we want to put this file in a package **world**
- Write a Java file as:

```
package world;
public class HelloWorld {
 public static void main(String[] args) {      System.out.println("Hello World");
  }
}
```

- Compile this file as

  File path> javac –d . HelloWorld.java

6.3 Multiple classes in Java file

- We can have only one public class in a single java file.
- Name of the file should be same as the name of public class
- In absence of public class, any class name can be given to the file name.

6.4 General Error (Setting Path)

'javac' is not recognized as an internal or external command, operable program or batch file

- When you get this error, you should conclude that your operating system cannot find the compiler (javac).
- Set Path of javac.exe so that OS can find it.
- To set the path go to advanced system settings after clicking the system option in control panel. Choose advanced tab, click on the environment variable button. Add user variable with variable name **path** and path of javac.exe file as value.
- The new path takes effect in each new command prompt window you open after setting the PATH variable.
- Now you can compile your Java program from any directory that contains your java file.

6.5 Java Access Modifiers

- Java supports four categories of accessibility rules
  - private
  - protected
  - public
  - default (not a keyword)

Modifiers can be used for class, member variables and member functions

6.6 How Access Modifiers affect a class?

- There can be only two possibilities, either class is a public class or just a class which means it is of default type.
- There can be only one public class in a single java file.

- The name of the java file must be the same as the name of the public class.
- Only public class can be accessed directly from outside the package.

}

6.7 How members of a class affected by access modifiers?

- When members of the class are **private**, they cannot access from outside the class body. They are meant to be accessed from the same class in which they are declared.
- When members are **protected**, they can be accessed from any class of the same package and child class from other package
- When members are **public**, they are accessible from any class of any package.
- When members are **default**, they are accessible only from the class of same package.

6.8 Java Constructors

- Constructor is an instance member function of a class
- The name of constructor is same as the name of the class.
- Constructor has no return type.
- Constructor can never be static
- A **constructor** is a special method that is used to **initialize a newly created object** and is called implicitly, just after the memory is allocated for the object
- It is **not mandatory** for the coder to write a constructor for the class. When there is no constructor defined in the class by programmer, compiler implicitly provide a default constructor for the class.
- Constructors can be parameterized, thus can be overloaded

Example: Write a class Box with length, breadth, height as private member variables and provide constructors to initialize objects.

```
public class Box
{
        private int length;
        private int breadth;
        private int height;
        public Box() {}
        public Box(int side)
        {
```

```
            length=side;
            breadth=side;
            height=side;
        }
        public Box(int l,int b,int h)
        {
            length=l;
            breadth=b;
            height=h;
        }
        public static void main(String[] args)
        {
            Box b1=new Box(12,10,5);
            Box b2=new Box();
            Box b3=new Box(5);
            b3.showDimension();
        }
        public void setDimension(int l,int b,int h)
        {
            length=l;
            breadth=b;
            height=h;
        }
        public void showDimension()
        {
```

```
            System.out.println("Length="+length);
            System.out.println("Breadth="+breadth);
            System.out.println("Height="+height);
        }
}
```

6.9 Initialization Block

- There are two types of initialization blocks
  - Instance Initialization Block
  - Static Initialization Block

Instance initialization block is unnamed block in the class. It executes just before the constructor. It is mainly used to initialize instance specific final members.
Static initialization block is unnamed block in the class, but qualified with static keyword. This block executes only once when class is first loaded. It is used to initialize static members and other onetime initialization task.

Type and execute the following code to understand the working flow of the program.

```
public class Test
{
 private int x;
 private static int k;

 {
  System.out.println("Initialization Block: x="+x);
  x=5;
 }
 static
 {
  System.out.println("Static Initialization Block: k="+k);
  k=10;
```

```
}

public Test()
{
 System.out.println("Constructor: x="+x);
}
public static void main(String []args)
{
  new Test();
  new Test();
}
}
```

Check Yourself:

Objective Type Question:

1. Which of these keywords is used to define packages in Java?

a) pkg

b) Pkg

c) package

d) Package

2. Which of these is a mechanism for naming and visibility control of a class and its content?

a) Object

b) Packages

c) Interfaces

d) None of the Mentioned.

3. Which of this access specifies can be used for a class so that its members can be accessed by a different class in the same package?

a) Public

b) Protected

c) No Modifier

d) All of the mentioned

4. Which of these access specifiers can be used for a class so that it's members can be accessed by a different class in the different package?

a) Public

b) Protected

c) Private

d) No Modifier

d) Import pkg.*

5. Which of the following is correct way of importing an entire package 'pkg'?

a) import pkg.

b) Import pkg.

c) import pkg.*

d) Import pkg.*

6. Which of the following is incorrect statement about packages?

a) Package defines a namespace in which classes are stored.

b) A package can contain other package within it.

c) Java uses file system directories to store packages.

d) A package can be renamed without renaming the directory in which the classes are stored.

7. Which of the following package stores all the standard java classes?

a) lang

b) java

c) util

d) java.packages

8. Which of the following statements are incorrect?

a) public members of class can be accessed by any code in the program.

b) private members of class can only be accessed by other members of the class.

c) private members of class can be inherited by a sub class, and become protected members in sub class.

d) protected members of a class can be inherited by a sub class, and become private members of the sub class.

9. What is the output of this program?

```
class access{
    public int x;
      private int y;
    void cal(int a, int b){
      x =  a + 1;
      y =  b;
    }
}
class access_specifier {
    public static void main(String args[])
    {
      access obj = new access();
      obj.cal(2, 3);
```

```
        System.out.println(obj.x + " " + obj.y);
    }
  }
```

a) 3 3

b) 2 3

c) Runtime Error

d) Compilation Error

10. What is the output of this program?

```
 class access{
      public int x;
        private int y;
      void cal(int a, int b){
        x =  a + 1;
        y =  b;
      }
void print() {
        system.out.println(" " + y);
      }
   }
   class access_specifier {
      public static void main(String args[])
      {
        access obj = new access();
        obj.cal(2, 3);
```

```
        System.out.println(obj.x);
        obj.print();
    }
  }
```

a) 2 3

b) 3 3

c) Runtime Error

d) Compilation Error

Subjective Type Qyestion:

1. WAP to create aclass Box having length, width,height as a data members. Initialize the data members using constructor.
2. WAP to define a class that represents Complex numbers with constructor to enable an object of this class to be initialized when it is declared and a default constructor when no argument is provided and define methods to do the following by passing objects as arguments
a) addition of two complex number
b) subtraction of two complex number
3. Create a package called shape containing an Abstract class called Figure with 2 data members for storing dimensions of the figure . Also provide appropriate methods for initializing and displaying values of the Figure.The class should also have an abstract method called area() which returns area of the Figure.

Now create an anotherpackage called myshape containing a class called Rectangle and a class called Circle.Both the classes should inherit Shape an override method area().

Finally create a package called useshape containing a class called TestShape having main() method .And Using dynamic method dispatch call the area() of both Circle and Rectangle Classes.

4. Write a JAVA Program to implement Inner class and demonstrate its Access Protections.

5. What do you mean by synchronized Non Access Modifier?
6. When to use private constructor?

Chapter 7: Inheritance

7.1 Object is a real world entity

Object in any object oriented language is used to represent any entity. Such mapping can be done by encapsulating all the properties and behaviors of the entity. A real world car has so many properties, like price, fuelType, colour, capacity, engine, etc. These property names are common to all the cars. So we define a class consist of all these property names as variables. We also provide methods to access these properties like setPrice(), setColour(), setFuelType(), etc. Instance of this class is capable to hold information of a car. We can create as many instances as required to represent different cars.

Later, a new requirement arises to represent a sports car. Sports car has all the features that a car has, but it has few more properties specific to sports car. It is obvious that sports car is a car so a sports car is a super set of car. If we define a separate class (SprotsCar) to meet the requirements, we redefine all the properties and methods of car, which is equivalent to unnecessary rework. This leads to increase in cost and time to develop software.

Another solution could be modifying the content of the car class. Add new properties and methods in the car class. Now we can create an instance that is capable to represent a sports car, but what about car, we cannot represent just a car.

Put above two solutions the garbage bag, as Java supports inheritance. We can define a class SportsCar specifying only properties and methods that are specific to the sports car, also

inheriting all the members of car class by using simple syntax. In this way, car class has its own identity and sports car has its own.

7.2 Inheritance

- Object-oriented programming allows classes to *inherit* commonly used state and behavior from other classes.
- In the Java programming language, each class is allowed to have one direct superclass, and each superclass has the potential for an unlimited number of *subclasses*
- Following is the syntax of inheritance:

  class SubClass extends SuperClass

  {

  }

- extends is a keyword
- Base class means Super Class
- Sub Class means Sub Class

Example: Define super class Person and sub-class Employee

```
class Person
{
 private String name;
 private int age;
 public void setName(String n)
 { name=n; }
 public void setAge(int a)
 { age=a; }
 public String getName()
 { return(name); }
 public int getAge()
 { return(age); }
}
```

```
class Employee extends Person
{
 private float salary;
 public void setSalary(float s)
 { salary=s;}
 public float getSalary()
 { return(salary); }
}
public class Example
{
 public static void main(String args[])
 {
  Employee e1=new Employee();
  e1.setName("Arun");
  e1.setAge(27);
  e1.setSalary(20000.0f);
  System.out.println("NAME: "+e1.getName());
  System.out.println("AGE: "+e1.getAge());
  System.out.println("SALARY: "+e1.getSalary());
 }
}
```

7.3 Transitive Relation

The inheritance relationship is transitive: if class x extends class y, then a class z, which extends class x, will also inherit from class y.

7.4 Is- a relationship

Java Inheritance supports ‘is-a’ relationship.

Example

Mango is a Fruit

Cat is an Animal

Student is a Person

Here ‘sub-class is a Super-class’.

7.5 Inheritance Rule

- Private members of the superclass are not accessible by the subclass and can only be indirectly accessed.
- Members that have default accessibility in the superclass are also not accessible by subclasses in other packages, as these members are only accessible by their simple names in subclasses within the same package as the superclass
- A subclass can extend only one superclass (no multiple inheritance)

7.6 Java Supports Three Kind of Inheritance

- Single Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance

7.7 Method Overloading

If a class has multiple methods by same name but different parameters, it is known as **Method Overloading**. Overloaded methods provide similar services but different implementations.

Unlike C++, java truly extends the scope of class in inheritance. Method overloading is possible even if the one version resides in superclass and another version resides in subclass.

Method overloading is a way to implement the concept of polymorphism.

Example

```
class Employee
{
  private int empid;
  private String ename;
```

```
  private float salary;
  public void searchEmployee( int empId)
  {
  //some code
  }
  public void searchEmployee(String ename)
 {
  //some code
 }
 public void searchEmployee(float salary)
  {
  //some code
  }
}
```

Compiler resolves the correct method invocation on the basis of arguments. This is called compile time polymorphism.

In java it is possible to have overloaded version in subclasses. Thus any subclass of Employee can specify a different version of searchEmployee() method with distinct arguments.

7.8 Method overriding

- Overriding of method is writing different implementation of the function whose prototype is same as the prototype of a function residing in superclass.
- Think about a class Car with shiftGear() method with specific implementation with respect to Car class. Now, consider a subclass SportsCar of class Car. SportsCar should also have shiftGear() service but with different implementation. The need can only be fulfilled by overriding the shiftGear() method with implementation specific to SportsCar.

- Whenever object of SportsCar invoke shiftGear() method, subclass's version should workout and when object of Car invoke shiftGear() method, superclass's version should workout. This is exactly happens in Java.
- Overridden methods in java are bind dynamically.
- Remember there is no virtual keyword in Java.

Example: displayMe() is the overridden method in the following example.

```
class Person
{
 private String name;
 private int age;
 public void displayMe()
 {
   System.out.println("Name: "+name);
   System.out.println("Age: "+age);
 }
 public void setName(String n)
 { name=n; }
 public void setAge(int a)
 { age=a; }
 public String getName()
 { return(name); }
 public int getAge()
 { return(age); }
}
class Employee extends Person
{
```

```
 private float salary;
 public void displayMe()
 {
   System.out.println(“Name: “+name);
   System.out.println(“Age: “+age);
   System.out.println(“Salary: “+salary);
 }
 public void setSalary(float s)
 { salary=s;}
 public float getSalary()
 { return(salary); }
}
public class Example
{
 public static void main(String args[])
 {
  Employee e1=new Employee();
  e1.setName("Arun");
  e1.setAge(27);
  e1.setSalary(20000.0f);
  System.out.println("NAME: "+e1.getName());
  System.out.println("AGE: "+e1.getAge());
  System.out.println("SALARY: "+e1.getSalary());
  e1.displayMe();
  Person p1= new Person();
```

```
 p1.displayMe();
 }
}
```

7.9 Comparison between overloading and overriding

Method overloading in Java occurs when two or more methods in the same class have the exact same name but different parameters (no of parameters are different or type of parameters are different)

Example:

1. Compiler error - can't overload based on the type returned - one method returns int, the other returns a float):

int changeDate(int Year) ;

float changeDate (int Year);

2. Compiler error - can't overload by changing just the name of the parameter (from Year to Month):

int changeDate(int Year);

int changeDate(int Month) ;

3. Valid case of overloading, since the methods have different number of parameters:

int changeDate(int Year, int Month) ;

int changeDate(int Year); 4. Also a valid case of overloading, since the parameters are of different types:

int changeDate(float Year) ;

int changeDate(int Year);

Overriding methods is completely different from overloading methods. If a derived class requires a different definition(body) for an inherited method, then that method can be redefined in the derived class. This would be considered overriding. An overridden method would have the **exact same method name, return type, number of parameters, and types**

of parameters as the method in the parent class, and the only difference would be the definition of the method.

Example of method overriding

```
public class Parent {

public int someMethod()

 {

System.out.println(“Parent Class”);

   return 1;

   }

}

public class Child extends Parent{

//Method Overriding:

 public int someMethod()

{

System.out.println(“Base Class”);

return 2;

 }

}
```

Check Yourself:

Objective Type Question:

1. Which of these keywords is used to refer to member of base class from a sub class?

a) upper

b) super

c) this

d) None of the mentioned

2. A class member declared protected becomes member of subclass of which type?

a) public member

b) private member

c) protected member

d) static member

3. Which of these is correct way of inheriting class A by class B?

a) class B + class A {}

b) class B inherits class A {}

c) class B extends A {}

d) class B extends class A {}

4. Which of the following statements are incorrect?

a) public members of class can be accessed by any code in the program.

b) private members of class can only be accessed by other members of the class.

c) private members of class can be inherited by a sub class, and become protected members in sub class.

d) protected members of a class can be inherited by a sub class, and become private members of the sub class.

5. What is the output of this program?

```
class A {
    int i;
    void display() {
        System.out.println(i);
```

```
    }
  }
  class B extends A {
    int j;
    void display() {
      System.out.println(j);
    }
  }
  class inheritance_demo {
    public static void main(String args[])
    {
      B obj = new B();
      obj.i=1;
      obj.j=2;
      obj.display();
    }
  }
```

a) 0

b) 1

c) 2

d) Compilation Error

6. What is the output of this program?

```
class A {
    int i;
}
class B extends A {
    int j;
    void display() {
        super.i = j + 1;
        System.out.println(j + " " + i);
    }
}
class inheritance {
    public static void main(String args[])
    {
        B obj = new B();
        obj.i=1;
        obj.j=2;
        obj.display();
    }
}
```

a) 2 2

b) 3 3

c) 2 3

d) 3 2

7. What is the output of this program?

```
class A {
    public int i;
    private int j;
}
class B extends A {
    void display() {
        super.j = super.i + 1;
        System.out.println(super.i + " " + super.j);
    }
}
class inheritance {
    public static void main(String args[])
    {
        B obj = new B();
        obj.i=1;
        obj.j=2;
        obj.display();
    } }
```

a) 2 2

b) 3 3

c) Runtime Error

d) Compilation Error

8. What is the output of this program?

```
class A {
    public int i;
    public int j;
    A() {
        i = 1;
        j = 2;
        }
}
class B extends A {
    int a;
    B() {
        super(ob);
    }
}
class super_use {
    public static void main(String args[])
    {
        B obj = new B();
          System.out.println(obj.i + " " + obj.j)
```

```
    }
  }
```

a) 1 2

b) 2 1

c) Runtime Error

d) Compilation Error

9. What is the output of this program?

```
  class A {
    public int i;
    protected int j;
  }
  class B extends A {
    int j;
    void display() {
      super.j = 3;
      System.out.println(i + " " + j);
    }
  }
 class Output {
    public static void main(String args[])
    {
      B obj = new B();
```

```
    obj.i=1;
        obj.j=2;
        obj.display();
      }
  }
```

a) 1 2

b) 2 1

c) 1 3

d) 3 1

10. Does a subclass inherit both member variables and methods?

a) No--only member variables are inherited.

b) No--only methods are inherited.

c) Yes--both are inherited.

d) Yes--but only one or the other are inherited.

Subjective Type Questive:

1.Create a class called Account and write a main method in a different class to briefly experiment with some instances of the Account class.

Using the Account class as a base class, write two derived classes called SavingsAccount and CurrentAccount. A SavingsAccount object, in addition to the attributes of an Account object, should have an interest variable and a method which adds interest to the account. A CurrentAccount object, in addition to the attributes of an Account object, should have an overdraft limit variable. Ensure that you have overridden methods of the Account class as necessary in both derived classes.

Now create a Bank class, an object of which contains an array of Account objects. Accounts in the array could be instances of the Account class, the SavingsAccount class, or the CurrentAccount class. Create some test accounts (some of each type).

Write an update method in the bank class. It iterates through each account, updating it in the following ways: Savings accounts get interest added (via the method you already wrote); CurrentAccounts get a letter sent if they are in overdraft.

The Bank class requires methods for opening and closing accounts, and for paying a dividend into each account.

Note that the balance of an account may only be modified through
the deposit(double) and withdraw(double) methods.

The Account class should not need to be modified at all.

2. Create a class called Employee whose objects are records for an employee. This class will be a derived class of the class Person class . An employee record has an employee's name (inherited from the class Person), an annual salary represented as a single value of type double, a year the employee started work as a single value of type int and a national insurance number, which is a value of type String.

Your class should have a reasonable number of constructors and accessor methods, as well as an equals method. Write another class containing a main method to fully test your class definition.

3) Write a program to create a class named shape. In this class we have three sub classes circle, triangle and square each class has two member function named draw () and erase (). Create these using polymorphism concepts.

4) Write a program to give a simple example for abstract class.

5) Write a program suppose, it is required to build a project consisting of a number of classes, possibly using a large number of programmers. It is necessary to make sure that every class from which all other classes in the project will be inherited. Since any new classes in the project must inherit from the base class, programmers are not free to create a different interface. Therefore, it can be guaranteed that all the classes in the project will respond to the same debugging commands.

6) Write a program to create interface A in this interface we have two method meth1 and meth2. Implements this interface in another class named MyClass.

7) Write a program to give example for multiple inheritance in Java.

8) Write a program to create interface named test. In this interface the member function is square. Implement this interface in arithmetic class.

Create one new class called ToTestInt in this class use the object of arithmetic class.

9) Create an outer class with a function display, again create another class inside the outer class named inner with a function called display and call the two functions in the main class.

10) Create class box and box3d. box3d is extended class of box. The above two classes going to pull fill following requirement :

a) Include constructor.

b) set value of length, breadth, height

c)Find out area and volume.

Note: Base class and sub classes have respective methods and instance variables.

Chapter 8: final, this and super keywords

And

Constructor in Inheritance

8.1 The keyword final

The keyword final can be used in the following four cases:

- final instance variable
- final local variable
- final class

- final methods

8.2 final instance variable

- A java variable can be declared using the keyword final. Then the final variable can be assigned only once.
- A variable that is declared as final and not initialized is called a blank final variable. Blank final variable forces either the constructors to initialize it or initialization block to do this job.

Example

```
class Account
{
 private double balance;
 private int accountNumber;
 private static final float rateOfInterest;
// initialization block
{
 rateOfInterest=3.5f;
}
…
}
```

8.3 final local variable

- Local variables that are final must be initialized before it's use, but you should remember this rule is applicable to non final local variables too.
- Once they are initialized, cannot be altered.

Example

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

```
 {
        final int x;
        System.out.println("x="+x); //Error: x is blank
        x=3;
        System.out.println("x="+x); //Correct
        x=5; //Error: can not modify
 }
}
```

8.4 final class

- Java classes declared as final cannot be extended. Restricting inheritance!

```
final class SavingAccount
{
 …
}
class SpecialAccount extends SavingAccount //Error: cannot extend final class
{
…
}
```

8.5 final methods

- Methods declared as final cannot be overridden

Example

```
class Person
{
```

```
 private String name;
 private int age;
 public final void displayMe()
 {
   System.out.println("Name: "+name);
   System.out.println("Age: "+age);
 }
 public void setName(String n)
 { name=n; }
 public void setAge(int a)
 { age=a; }
 public String getName()
 { return(name); }
 public int getAge()
 { return(age); }
}
class Employee extends Person
{
 private float salary;
 public void displayMe() //Error: Cannot Override
 {
   System.out.println("Name: "+name);
   System.out.println("Age: "+age);
   System.out.println("Salary: "+salary);
 }
```

```
 public void setSalary(float s)
 { salary=s;}
 public float getSalary()
 { return(salary); }
}
public class Example
{
 public static void main(String args[])
 {
  Employee e1=new Employee();
  e1.setName("Arun");
  e1.setAge(27);
  e1.setSalary(20000.0f);
  System.out.println("NAME: "+e1.getName());
  System.out.println("AGE: "+e1.getAge());
  System.out.println("SALARY: "+e1.getSalary());
  e1.displayMe();
  Person p1= new Person();
  p1.displayMe();
 }
}
```

8.6 The keyword this

- The ***this*** object reference is a local variable in instance member methods referring the caller object
- ***this*** keyword is used as a reference to the current object which is an instance of the current class

- The ***this*** reference to the current object is useful in situations where a local variable hides, or shadows, a field with the same name.
- If a method needs to pass the current object to another method, it can do so using the ***this*** reference.
- Note that the ***this*** reference cannot be used in a static context, as static code is not executed in the context of any object

Example

```
class Box
{
  private int length,breadth,height;
  public void setDimension(int length, int breadth, int height)
  {
        this.length=length;
        this.breadth=breadth;
        this.height=height;
  }
 ...
}
```

Above example shows, how this object reference can be used to resolve name conflict issue between local variables and instance variables. The keyword this is used to refer calling object.

8.7 The keyword super

- The keyword super also references the current object, but as an instance of the current class's super class.
- The keyword super can be used to resolve name conflict between superclass and subclass.
- super is available only instance member methods of child class

Example

Example

```
class Person
{
 private String name;
 private int age;
 public void displayMe()
 {
   System.out.println("Name: "+name);
   System.out.println("Age: "+age);
 }
 public void setName(String n)
 { name=n; }
 public void setAge(int a)
 { age=a; }
 public String getName()
 { return(name); }
 public int getAge()
 { return(age); }
}
class Employee extends Person
{
 private float salary;
 public void displayMe()
 {
   super.displayMe(); //use of super keyword
   System.out.println("Salary: "+salary);
```

```
 }
 public void setSalary(float s)
 { salary=s;}
 public float getSalary()
 { return(salary); }
}
public class Example
{
 public static void main(String args[])
 {
  Employee e1=new Employee();
  e1.setName("Arun");
  e1.setAge(27);
  e1.setSalary(20000.0f);
  System.out.println("NAME: "+e1.getName());
  System.out.println("AGE: "+e1.getAge());
  System.out.println("SALARY: "+e1.getSalary());
  e1.displayMe();
  Person p1= new Person();
  p1.displayMe();
 }
}
```

8.8 Constructor in Inheritance

- Constructors are not inherited by subclass
- Sub class's constructor invokes constructor of super class.

- Explicit call to the super class constructor from sub class's constructor can be made using super().
- You can write a subclass constructor that invokes the constructor of the superclass, either implicitly or by using the keyword super.
- Invocation of superclass constructor when explicitly mentioned must come as the first line in the code block.

Example

```
class Person
{
 private String name;
 private int age;
public Person()
{
    System.out.println("Constructor of Person");
}

}
class Employee extends Person
{
 private float salary;
 public Employee()
{
    System.out.println("Constructor of Employee");
}
}
public class Example
{
 public static void main(String args[])
 {
    Employee e1=new Employee();
 }
}
```

Output

Constructor of Person

Constructor of Employee

Example: Explicit call to super class constructor

```
class Person
{
 private String name;
```

```
 private int age;
public Person()
{
    System.out.println("Constructor of Person");
}

}
class Employee extends Person
{
 private float salary;
 public Employee()
{
    super();
    System.out.println("Constructor of Employee");
}
}
public class Example
{
 public static void main(String args[])
 {
    Employee e1=new Employee();
 }
}
```

Output

Constructor of Person

Constructor of Employee

8.9 Constructor Chaining

- Constructor can call other constructors of the same class of superclass
- Constructor call from a constructor should be the first step. (call should appear in the first line)
- Such series of invocation of constructors is known as constructor chaining.

Example

```
class Person
{
 private String name;
 private int age;
```

```
public Person()
{
    this(25);
    System.out.println("First Constructor of Person");
}
public Person(int a)
{
    this("Rahul");
    age=a;
    System.out.println("Second Constructor of Person");
}
 public Person(String n)
{
    name=n;
    System.out.println("Third Constructor of Person");
}

}
class Employee extends Person
{
 private float salary;
 public Employee()
{
    this(1000.0f);
    System.out.println("First Constructor of Employee");
}
public Employee(float s)
{
    super();
    salary=s;
    System.out.println("Second Constructor of Employee");
}

}
public class Example
{
 public static void main(String args[])
 {
    Employee e1=new Employee();
 }
}
```

Output

Third Constructor of Person

Second Constructor of Person

First Constructor of Person

Second Constructor of Employee

First Constructor of Employee

Check Yourself:

Objective Type Question:

1) Which method can't be inherited(it is possible to have more than one right ans).

a)private method

b)constructor

c)destructor

d)protected method

2) Which keyword is used to call base class constructor.

a) super

b) this

c) name of constructor it self

d) all true

3)which keyword is used to call constructor of own class from constructor it self.

a) super

b) this

c) name of constructor

d) all true

4) If only parameterized constructor is available in base class can it possible for child class constructor to call it implicitly?

a) yes

b) no

5) Consider the following code

```
 class A
{
  A()
{
}
}
class B extends A
{
 B()
{
 System.out.println("welcome");
 super();
}
}
```

Now you create a object of B class under another class within main method. What will happen?

a)exception

b)everything is right

c)error

d)none

6) If you have "n" parameterized constructor in base class which version of constructor gets called implicitly by child class constructor?

a)no one constructor will call

b)code generate syntax error

c)no one constructor will call but object of child class can be created.

d)exception

7) If a constructor is private you can't inherit the class.

a) true

b) false

8) If a constructor is protected you can't inherit the class.

a) true

b) false

9) Can u make constructor static?

a) yes

b) no

10) What will happen when you mention return type with constructor?

a) error

b) exception

c) java automatically removes the return type and code will runs normally

d) none

Subjective Type Question:

1) What is the difference between this and super keyword? Explain with example.

2)What is the use of final keyword in java?

3) Can values of variables defined within a final class, be changed after instantiation? and if those variables declared as final, then?

4) What is the order of constructor call in multilevel inheritance. Explain with example.

5) What does super keyword do?

6) Can a constructor be made final?

Chapter 9:

Abstract Class and Interface

9.1 Abstract Class

- **Java Abstract classes** are used to declare common characteristics of subclasses.
- An abstract class cannot be instantiated.
- Abstract classes are declared with the abstract keyword.

Example:

```
abstract class Person{
    private String name;
    private int age;
    public void setName(String n)  { name=n; }
    public void setAge(int a)  { age=a; }
}

class AbstractExample1{
    public static void main(String[] args)
    { Person p=new Person(); }  //can't instantiated
}
```

- It can only be used as a superclass for other classes that extend the abstract class.
- Like any other class, an abstract class can contain fields that describe the characteristics and methods that describe the actions that a class can perform.

Example

```
abstract class Person
{
```

```
        private String name;
        private int age;
        public void setName(String n)
        { name=n; }
        public void setAge(int a)
        { age=a; }
        public String getName()
       {return(name);}
       public int getAge()
       {return(age);}
}
class Student extends Person
{
      private int rollno;
      public void setRollno(int r)
      { rollno=r;}
      public void showStudent()
      {  System.out.println("Roll no:"+rollno+"Name: "+getName()+"Age:  "+getAge());}
}
 class AbstractExample2
 {
      public static void main(String[] args)
      {
                Student s=new Student();
                s.setName("Amar");
                s.setAge(18);
                s.setRollno(100);
                s.showStudent();
      }
}
```

9.2 Abstract Method

- An abstract class can include methods that contain no implementation. These are called abstract methods. The abstract method declaration must then end with a semicolon rather than a block.
- If a class has any abstract methods, whether declared or inherited, the entire class must be declared abstract

Observe the following Example

```
class Person{
 …
 abstract void show();
}
class Student extends Person{
 …
```

```
}
 class AbstractExample3{
  public static void main(String[] args){
     Student s=new Student();
  }
}
```

- Code will generate an error, as the Person class is not specified abstract. Whenever a class contains abstract method it is mandatory to make the class abstract. An object must have an implementation for all of its methods. You need to create a subclass that provides an implementation for the abstract method.

Example

```
abstract class Person
{
    private String name;
    public void setName(String n)
    { name=n; }
    public String getName()
    {return(name);}
    abstract void show();
}
class Student extends Person
{
    private int rollno;
    public void setRollno(int r)
    { rollno=r;}
    public void show()
    {  System.out.println("Roll no:"+rollno+" Name: "+getName()); }
}
class AbstractExample4
{
    public static void main(String[] args)
    {
            Student s=new Student();
            s.setName("Amar");
            s.setRollno(100);
            s.show();
    }
}
```

9.3 Interface

- **Interface definition begins with a keyword interface.**

- An interface like that of an abstract class cannot be instantiated.
- Interfaces just specify the method declaration (implicitly public and abstract) and can only contain fields (which are implicitly public static final).
-

Example:

```
interface Operations{
 int a=5, b=4;
 int sum();
 int diff();
}
class Calculator implements Operations{
   public int sum(){ return(a+b); }
   public int diff(){ return(a-b); }
}
class InterfaceExample1{
 public static void main(String[] args){
  Calculator c=new Calculator();
  System.out.println("Sum is: "+c.sum()+"Diff is "+c.diff());
 }
}
```

- Interface do not have constructors.

- If a class that implements an interface does not define all the methods of the interface, then it must be declared abstract and the method definitions must be provided by the subclass that extends the abstract class.

Example

```
interface Admission

{

 int registration();

 int batchAllotment();

 int iCardGeneration();

}
```

Example

```
interface GateKeeper
{
 int authentication();
 int alarm();
 int securitySetting();
}
```

9.4 Multiple Extensions of Interfaces

- Multiple extension is allowed when extending interfaces i.e. one interface can extend none, one or more interfaces.
- If a class has any abstract methods, whether declared or inherited, the entire class must be declared abstract

Example

```
interface I1
{
    void f1();
    void f2();
}
interface I2
{
    void f3();
    void f4();
}

interface I3 extends I1, I2{
    void f5();
}
```

9.5 Object Reference of Interface

- You cannot create object of any interface but creation of object reference is possible.
- Object reference of interface can refer to any its subclass type.
- When Object reference of an interface is referring to an object of the class which implements the interface, it can only access members originally belong to interface.

9.6 Interface and Abstract Class

- Abstract class can have any access modifiers for members. Interface can have only public members
- Abstract class may or may not contain abstract method. Interface can not have defined method.
- Abstract class can have static or non static members. Interface can have only static member variables.
- Abstract class can have final or non final members. Interface can have only final member variables.

Check Yourself:

Objective Type Question:

1. Variables of an interface are by default

a) final

b) static.

c) public.

d) all of the above.

2. Choose the correct one:

a) interfaces help to implement multilevel inheritance in java.

b) interfaces help to implement multiple inheritance in java.

c) a class cannot implement more than one interface.

d) interfaces must have concrete methods.

3. A class implements an interface but does not override all methods then

a) it should be declared as final class.

b) it should be declared as static class.

c) it should be declared as abstract class.

d) it should be both declared final and static.

4. An interface

a) contains method definitions.

b) cannot be extended to create another interface.

c) method must be declared as private.

d) can have reference.

5. Consider the following code:

```
interface Area
{
public void ar(int a);
}
class Sq implements Area
{
  int len;
public void area(int a)
{
 len=a;
System.out.println("Area:"+ len*len);
}
class D
{
 public static void main(String args[])
{
  //insert code here
}
```

Replace the "//insert code here " with appropriate option:

a) Sq s=new Sq();

b) Area a=new Area();

c) Area s1=new Sq();

d) A and C both.

6. Choose the appropriate option:

a. Data members of an interface are by default final.

b. An abstract class has implementations of all methods defined inside it.

a) a-false, b-false

b) a-false, b-true

c) a-true, b-false

d) a-true, b-true

7. Completely blank interfaces are called as:

a) Blank interface.

b) Markup interface.

c) Tagging interface.

d) Zero level interface.

8. Consider the following code:

```
interface Arr
{
  public void show(String  s);
}
```

a. public abstract class Fr implements Arr

{

```
  public void show(String  j){ }
}
```

b. public abstract class Fr implements Arr { }

c. public class Fr extends Arr

```
{
    public void show(String  j) { }
```

d. public class Fr implements Arr

```
   {
    public void show(String  j) { }
    public void show(int  a) { }
     }
```

Which one is the correct option regarding the given code.

a) a

b) d

c) b and d.

d) a and c.

9. The concept of multiple inheritance is implemented in Java by:

a. extending two or more classes.

b. extending one class and implementing one or more interfaces.

c. implementing two or more interfaces.

Choose the correct option:

a) b

b) a

c) b and c.

d) a and b.

10. Which one is correct?

a) public final int a=4;

b) public int a=4;

c) static int a=4;

d) public static final int a=4;

Subjective Type Question:

1) Write a program to demonstrate actual use of interface. For this make an interface called Animal inside this make a method eat(). Now construct two derived class Cow and Lion override the method in appropriate way and generate the polymorphic call for eat() of Lion and Cow class.

2)Make an interface called JavaTech inside this make a method working() now make four child class JSE, JEE, JME and JavaCard override working() method and display working of each technology inside their working() method.

3)Construct an interface called Shape and inside it make a method called area() now make derived classes: Rectangle, circle, Square and Triangle now override area() method in appropriate way.

4)Design an interface Car and make methods maxSpeed(), fuelType(), engineCC(). now make Safari,Fortuner,Civic as derived class and override the method in appropriate way.

5)Design an interface called CellPhone and make methods availabeOS(),cameraPixel(),

soundType(),isGpsEnabled() now make SonyExperia_c ,IPhone_5, NokiaLumia as derived class override the method in appropriate way

6) Can abstract class have constructors in Java?

7) Can abstract class implements interface in Java? Does they require to implement all methods?

8) Can abstract class be final in Java?

9) When do you favor abstract class over interface?

10) What must a class do to implement an interface?

Chapter 10:

Taking Input from Keyboard

10.1 Scanner Class

- We can read java input from system.in using Scanner class
- Scanner is final class, that is cannot be extended.
- A Scanner breaks its input into tokens using a delimiter pattern, which by default matches whitespace
- The resulting tokens may then be converted into values of different types using the various next methods.
- Scanner class is a part of java.util package

Example:

```
import java.util.*;
class InputExample1{
 public static void main(String []args){
  System.out.println("Enter your name and age:");
  Scanner sc=new Scanner(System.in);
  String name=sc.next();
  int age=sc.nextInt();
  System.out.println("Name: "+name+" Age: "+age);
 }
}
```

10.2 Useful Methods of Scanner Class

Following methods are used to get filtered data from input buffer

- next()
- nextLine()
- nextBoolean()
- nextByte()
- nextDouble()
- nextFloat()

- nextInt()
- nextLong()
- nextShort()

Following Methods are used to check whether the buffered data is convertible to specific type or not. They return boolean type value.

- hasNext()
- hasNextLine()
- hasNextBoolean()
- hasNextByte()
- hasNextDouble()
- hasNextFloat()
- hasNextInt()
- hasNextLong()
- hasNextShort()

Check Yourself:

Objective Type Question:

1. We Use System.in with Scanner class to take input what is “in”?

a) Object of InputStream class.

b) Object of OutputStream class

c) A Constant Variable.

d) All are false.

2. Choose correct one for this statement “Scanner class can take input from only keyboard”
a) True
b) It depends on argument passed to scanner constructor.
c) False
d) none

3. When you use the nextXxx() methods to take input and you pass the wrong input ,which exception they will generate?
a) InputMismatchException
b) NumberFormatException
c) IOException
d) None of these

4. If you want to take String which contain spaces as input which nextXxx() method u will prefer?
a) Next()
b) nextLine()
c) has Next();
d) None of these

5. All nextXxx() methods are ?
a) Static and final
b) Nonstatic but final
c) Only nonstatic
d) Only static

6. When u write Scanner s=new Scanner(System.in); Which type of constructor java will call?
a) Scanner(InputStream)
b) Scanner(OutputStream)
c) Scanner(Reader)
d) Scanner(String)
7. Use the next() method to take input and give the input ”Hello world” what will happen when you print it ?
a) Hello
b) Hello World
c) Exception
d) Nothing will display

8. Scanner class can’t be inherited because
a) It has private constructor
b) Java does not allow
c) It is final class
d) It is private class

9. Scanner class have total ____ contructors?
a) 1
b) 2
c) 8
d) 7

10. All hasNextXxx() method has a return type?
a) Void
b) String
c) Each hasNextXxx() method has different return type
d) boolean

Subjective Type Question:

1) Write a program to accept two integer from the user do the below operation on it:

a) Addition

b) Subtraction

c) Division

d) Multiplication

2) Write a program to accept thre integer value and find out there sum and average.

3) Write a program which uses all the nextXXX() method?

4) Write a program to read the data from file using scanner?

5) Write a program to accept string, float and integer value from the user and display according.

6) Write a program to accept 4 integers from user and find greatest number?

7) Write a program which demonstrate that method nextLine() also used for cleaning the buffer?

8) Write a program which accepts number in String format using scanner then convert all the numbers in integer and display their sum?

9) Write a program which demonstrate that when u give the wrong input to nextXXX() method it generates InputMismatchException?

10) Write a program which accept name, father name, mother name, college, semester, enrollment no and percentage of previous semester and display the below output like this.

Your name is : sumit

Your father name is : shri Ashok

Prepared By: Ankit Desai

Chapter 11: Arrays

11.1 Declaring Arrays in Java

- int arr[5]; //error: Cannot declare array as you did in C/C++
- int arr[]; //This is not an array, but it is a reference variable capable to refer an array
- int [] arr = new int[5]; // Correct way of creating array of 5 int blocks.
- In Java array is also created with the help of new keyword.
- int arr[] =new int[5]; //Also correct with the same meaning
- int arr[]=new int[] {2, 4, 6, 8, 10}; // Creating array of 5 int blocks with specific initial values
- int arr[]=new int[2] {2, 4, 6, 8, 10}; //error
- int arr[]=new int[5]{2,3,4,5,8}; //error: you can not mention size and values together
- int arr[]={2, 4, 6, 8, 10}; //Also correct. You can create an array without using new keyword.

- Since array is always dynamically created, it can never be blank.
  int []arr=new int[3];
  System.out.println("arr[0]="+arr[0]);

  Above statement print 0.

  Example: Find Error in below program:

```
public class ArrayExample
{
 public static void main(String args[])
 {
  int arr[];
  arr[0]=25;
  arr[1] =50;
 }
}
```

In the above example, we have created array reference with the name arr but not created any array. Since arr is a local variable it contains nothing that is blank variable and it is illegal to access variable without initialization

11.2 Initialization after declaration

You can initialize an array after declaring it. For example

```
public class ArrayExample
{
 public static void main(String args[])
 {
  int arr[];
  arr = new int[3];
  arr[0]=10;
  arr[1]=20;
  arr[2]=30;
 }
}
```

Example: Write a program to input 5 numbers from the user and store them in an array. Also print all five values on the console.

```
import java.util.*;
class ArrayExample1{
 public static void main(String[] args)
{
   int []arr=new int[5];
   Scanner sc=new Scanner(System.in);
   System.out.println("Enter 5 numbers");
   for(int i=0;i<5;i++)
            arr[i]=sc.nextInt();

   for(int i=0;i<5;i++)
            System.out.println("arr["+i+"]="+arr[i]);
 }
}
```

11.3 Two dimensional array declaration

- int [][]a=new int[4][5]; //Correct way to create a two dimensional array
- int [][]a=new int[][]; //Error: cannot declare without mentioning length of both the dimensions
- int [][]a=new int[][5]; //Error: cannot leave first bracket empty
- int [][]a=new int[4][]; //Correct: can leave second bracket empty

- int [][]a=new int[][] {{3,4},{5,6},{7,8}}; //Correct way to assign value during declaration

Example : Find Error in below program:

```
public class Example1
{
 public static void main(String []args)
 {
        int [][]a=new int[4][];
        System.out.println("a[0][0]"+a[0][0]);
 }
}
```

Output

No compile time error

But there an error at run time : NullPointerException

Example: Find Output in below program:

```
public class Example1
{
 public static void main(String []args)
 {
        int [][]a=new int[4][];
        System.out.println("a[0]="+a[0]);
 }
}
```

Output

a[0]=null

11.4 Using length property of an array:

```
public class Example1
{
 public static void main(String []args)
 {
        int [][]a=new int[4][];
        System.out.println("Length="+a.length);
 }
}
```

Output

Length=4

Example: Find Output for the below program:

```
public class Example1
{
 public static void main(String []args)
 {
        int [][]a=new int[4][];
        System.out.println("Length="+a[0].length);
 }
}
```

Output:

Run Time Error: NullPointerException

Since a[0] contains null, that is, not pointing to any list of blocks, so it is invalid to use length property

11.5 Array In Memory

int [][]a=new int[4][5];

Example: Find Output of the below program:

```
public class Example1
{
 public static void main(String []args)
 {
        int [][]a=new int[4][5];
        System.out.println("Length="+a[0].length);
 }
}
```

Output

Length=5

11.6 Distinct Length in 2D Array

In java it is possible to declare a 2d array with different length of each array.

Example

```
public class Example1

{

 public static void main(String []args)

 {

        int [][]a=new int[4][];

        a[0]=new int[5];

        a[1]=new int[3];

        System.out.println("Length="+a[1].length);

 }
```

}

Output

Length = 3

Check Yourself:

Objective Type Question:

1.What does the following code will do?

int a[]={1,2,3};

int b[]={1,2,3};

System.out.println(a==b);

a) Will not compile.

b) Display false.

c) Generate Exception.

d) Will print nothing.

2. Which one of the following is a valid statement?

a) char c[]= new char();

b) char c[]= new char[3];

c) char c[]= new char(4);

d) char c[]= new char[];

3. Consider the following

a) int b [3]={7,9,3,4};

b) int b[3]={4,8,6};

c) int b[]={3,7,6};

Choose the correct one:

a) a is wrong.

b) a & b both are wrong.

c) a & c are correct.

d) None of the above.

4. What will be the output?

```
 public static void main(String args[])
{
  int x[]=new int [3];
  System.out.println(x[0]);
}
```

a) Compile time error.

b) Display 0

c) Exception

d) Nothing will be displayed.

5. Which of these is an incorrect Statement?

a) It is necessary to use new operator to initialize an array.

b) Array can be initialized using comma separated expressions surrounded by curly braces.

c) Array can be initialized when they are declared.

d) None.

6. Find the correct one

a) int a[][]=new int[3][3]{10,20,30};

b)int arr[][]=new int arr[2][4];

c) int a[][] = new int []{{2,3,4}{4,8,7}};

d) int a[][]={{10,20,30},{3,5,8,7},{4,1,2,5,7}};

7. What will be the output ?

```
int x[][] = {{0,1,2,3,4},{0,1,2},{0,1,2,3}};

System.out.println(x[0].length);
```

a) 3

b) 5

c)0

d) 2

8. Consider the following code:

```
int x[][];
```

x= new [3][4];

Choose the wrong one:

a) System.out.println(arr.length);

b) System.out.println(arr[0].length);

c) System.out.println(arr[0][0].length);

d) System.out.println(arr[1].length);

9. Which of the following correctly initializes an array arr to contain four elements each with value 0?

a) int[]arr = {0,0,0,0};

b) int[]arr = new int[4];

c)
```
int[]arr = new int[4];
for (int i= 0; i<arr.length; i++)
arr[i] = 0;
```

A. a only
B. b only
C. a & b only
D. a, b & c.

10. Which of these is an incorrect array declaration?

a) int arr[] = new int[5];

b) int [] arr = new int[5];

c) int arr[];

arr = new int[5];

d) int arr[] = int [5] new;

Subjective Type Question:

1. Write a program to accept a 2 dimensional array and display the transpose of that matrix.

2. Write a program to find largest element in an array.

3. Write a program to arrange a set of n integers in descending order.

4. Write a program to create two dimensional array of size 3x4 size and print the sum of each row.

5. Write a program to create a jagged array of 2 rows with user defined column size and display all the values along with sum(rows).

6. Write a program to accept integers via command line arguments and display their sum and average.

7. Write a program to merge two arrays.

8. Write a program for salesman performance analysis. Ask the user to input how many salesman are there , then ask the user about the number of sales made by every salesman. Finally display average sales made by each salesman.

9. Write a program to accept user defined two 2 dimensional array and display their sum.

10. Write a program to perform binary search in an array.

Chapter 12: Object Class in Java

12.1 Object Class

- Class Object is the root of the class hierarchy
- Every class has Object as a direct or indirect superclass
- Object class provides several methods, some of them are
  - clone()
  - equals()
  - finalize()
  - toString()

12.2 Brief description about methods of Object Class

Method	Description
clone()	Creates and returns a copy of this object.
equals(Object obj)	Indicates whether some other object is "equal to" this one.
finalize()	Called by the garbage collector on an object when garbage collection determines that there are no more references to the object
toString()	Returns a string representation of the object

- A subclass overrides the finalize method to dispose of system resources or to perform other cleanup.
- The finalize method of class Object performs no special action; it simply returns normally. Subclasses of Object may override this definition.
- Finalize method is invoked implicitly on object's destruction

Observe the following Java Code

```
class ComplexNumber{
 private int a,b;
 public ComplexNumber() { a=0; b=0;}
 public ComplexNumber(int x,int y){ a=x; b=y;}
}
class StringExample3{
 public static void main(String[] args){
  ComplexNumber cn=new ComplexNumber(3,4);
  System.out.println("Z= "+cn);
 }
}
```

Explanation:

- In the previous example, when we try to print ComplexNumber's object (**cn**), by default **toString()** method of Object class is invoked and return a string containing class name, '@' symbol and hash code in hex format.
- To make it meaningful we need to override toString() method. Same is illustrated in next example

Modified Example

```
class ComplexNumber{
 private int a,b;
 public ComplexNumber() { a=0; b=0;}
 public ComplexNumber(int x,int y){ a=x; b=y;}
 public String toString(){  return(a+"+"+b+"i");  }
}
class StringExample3{
 public static void main(String[] args){
  ComplexNumber cn=new ComplexNumber(3,4);
  System.out.println("Z= "+cn);
 }
}
```

Check Yourself:

Objective Type Question:

1) Which class is the base class of every class.

a) lang

b) Object

c) Java

d) none

2) Object class available in which package?

a) util

b) java

c) lang

d) none

3) How many constructors java class has by default?

a) 1

b) 2

c) 5

d) 3

4) If class A is base of Class B and we know by default every class in java inherit the object class so which type of inheritance Class B will show

a) single

b) multiple

c) multilevel

d) multipath

5) Object class can’t extend any class

a) true

b) false

6) Is it allow to accept object of any class in Reference of Object class

a) yes

b) no

7) toString() is abstract method.

a) yes

b) no

8) It is not possible for java(sun) developers to make even one method as abstract in Object class

a) yes

b) no

Subjective Type Question:

1. Why is it so Object class is base class of every class?

2. Why it happens that when we pass a String object reference to the method println() then we don't get hashcode but get content of string.

3. Give the list of Java Object class method

.

Prepared By: Ankit Desai

Chapter 13: Strings

13.1 String Class

- A **java.lang.String class** is final which implies no class can extend it
- **Java String Class** is immutable, i.e. **Strings in java,** once created and initialized, cannot be changed on the same reference.
- The java.lang.String class differs from other classes, one difference being that the String objects can be used with the += and + operators for concatenation
- A simple String can be created using a string literal enclosed inside double quotes as shown
- String str1 = “My name is Bond”;
- If two or more Strings have the same set of characters in the same sequence then they share the same reference in memory
- String str1 = “My name is Bond”;
  String str2 = “My name is Bond”;
  String str3 = “My name ”+ “is Bond”;

13.2 In Memory

- All the String references str1, str2 and str3 denote the same String object

- String str4 = “Bond”;
  String str5 = “My name is ”+ str4;

  String str6 = “My name is Bond”;

13.3 Creating String with new keyword

- String str5 = new String(“My name is Bond”);
- Creating string with new keyword always allocates a new memory block for object

13.4 Comparing two Strings

```
class StringExample1{

 public static void main(String[] args){

  String s1="computer";

  String s2="computer";

  String s3=new String("computer");

  System.out.println("Result 1:"+(s1==s2));

  System.out.println("Result 2:"+s1.equals(s2));

  System.out.println("Result 3:"+(s1==s3));

  System.out.println("Result 4:"+s1.equals(s3));

 }

}
```

Output

Result 1: true

Result 2: true

Result 3: false

Result 4: true

- Comparing two string can be done with the help of equals() method of string class
- The == operator is used when we have to compare the String object references

- If two String variables point to the same object in memory, the comparison returns true. Otherwise, the comparison returns false

13.5 Calculating length of the string

```
class StringExample2{
 public static void main(String[] args){
  String s1="computer";
  System.out.println("Length is "+s1.length()); //Length is 8
 }
}
```

13.6 List of useful methods of String Class

- toUpperCase()
- toLowerCase()
- equalsIgnoreCase(String anotherString)
- indexOf(int ch)
- **indexOf**(int ch, int fromIndex)
- indexOf(String str)
- **indexOf**(String str, int fromIndex)
- **lastIndexOf**(int ch)
- **lastIndexOf**(int ch, int fromIndex)
- **lastIndexOf**(String str)
- **lastIndexOf**(String str, int fromIndex)
- **charAt**(int index)
- **substring**(int beginIndex)
- **substring**(int beginIndex, int endIndex)
- **replace**(char oldChar, char newChar)
- **trim**()
- toCharArray()
- toString()
- compareTo(String s)

Find Output of the following code

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

```
 String s1="Computer";
 String s2="cOmPuTeR";
 System.out.println("Comparision: "+(s1==s2));
 System.out.println("Comparision: "+s1.equals(s2));
 System.out.println("Comparision: "+s1.compareTo(s2));
 System.out.println("Comparision: "+s1.equalsIgnoreCase(s2));
 System.out.println("Conversion: "+s1.toUpperCase());
 System.out.println("Conversion: "+s2.toLowerCase());
 System.out.println("Index of: "+s1.indexOf('m'));
 System.out.println("Index of: "+s1.indexOf('m',2));
 System.out.println("Index of: "+s1.indexOf("ter"));
 System.out.println("Character At: "+s1.charAt(4));
 System.out.println("Sub-String: "+s1.substring(2));
 System.out.println("Sub-String: "+s1.substring(2,5));
 System.out.println("Replace: "+s1.replace('m','M'));
}}
```

13.7 StringBuffer class

- StringBuffer class is a mutable class unlike the String class which is immutable
- String buffers are preferred when heavy modification of character strings is involved (appending, inserting, deleting, modifying etc).

Find Error

```
class StringExample4{
 public static void main(String[] args){
  StringBuffer sb1="hello";
  System.out.println(sb1);
 }
}
```

Error: Incompatible types. In above program we are trying to assign a string type value to StringBuffer, which is invalid.

Correct Way

```
class StringExample4{
 public static void main(String[] args){
  StringBuffer sb1=new StringBuffer("hello");
  System.out.println(sb1);
 }
}
```

13.8 Appending String to StringBuffer

```
class StringExample4{
 public static void main(String[] args){
  StringBuffer sb1=new StringBuffer();
  sb1.append("hello");
  sb1.append(" Students");
  System.out.println(sb1);
 }
}
```

In the above example, append method is used to append string to existing StringBuffer object.

13.9 Deleting Content from StringBuffer

```
class StringExample5{
 public static void main(String[] args){
  StringBuffer sb1=new StringBuffer("Hello Students");
  System.out.println(sb1);
  sb1.delete(6,sb1.length());
  System.out.println(sb1);
 }
}
```

In the above example, delete() method of StringBuffer is used to delete a substring from the String.

13.10 Comparing two StringBuffers

```
class StringExample6{
 public static void main(String[] args){
  StringBuffer sb1=new StringBuffer("computer");
  StringBuffer sb2=new StringBuffer("computer");

  System.out.println("sb1==sb2:"+(sb1==sb2));
 }
}
```

In this example, we are not comparing strings, rather we are comparing references.
Strings can be obtained from string buffers. Since the StringBuffer class does not override the equals() method from the Object class, contents of string buffers should be converted to String objects for string comparison. The correct way to compare two StringBuffer objects is as follows:

```
class StringExample6{
 public static void main(String[] args){
```

```
  StringBuffer sb1=new StringBuffer("computer");
  StringBuffer sb2=new StringBuffer("computer");
System.out.println(sb1.toString().equals(sb2.toString()));
 }
}
```

13.11 List of useful methods of StringBuffer

- **length**()
- **charAt**(int index)
- **setCharAt**(int index, char ch)
- **toString**()
- **delete**(int start, int end)
- **replace**(int start, int end, String str)
- **reverse**()
- **append**(String str)

Check Yourself:

Objective Type Question:

1. String class comes from which of the following package?

A. java.util;

B. java.io;

C. java.lang;

D. java.Object;

2. Consider the given piece of code

```
  String name1=”ABC”;

  String name2=”ABC”
```

How many references and objects will be created for the above code?

A. 1 reference, 2 objects.

B. 2 references, 1 object.

C. 1 reference, 1 object.

D. 2 references, 2 objects.

3. Consider the given piece of code

String name1= new String(“ABC”);

String name2=new String(“ABC”);

How many references and objects will be created for the above code?

A. 1 reference, 2 objects.

B. 2 references, 1 object.

C. 1 reference, 1 object.

D. 2 references, 2 objects.

4. String str=””;

What will be the output of the following code:

System.out.println(str.length);

A. 0

B. 1

C. null.

D. undefined.

5. Now assume that we have created the String reference at class level as:

String str=””;

What will be the output of the following:

System.out.println(str.length);

A. 0

B. 1

C. Null pointer exception.

D. Undefined.

6. How many overloaded versions of the method indexOf() are there in java?

A. 4

B. 3

C. 2

D. 0

7. String Objects are by default ..

A. mutable

B. immutable.

C. both A and B.

D. none of the above.

8. In Q2. if we compare the Strings as:

if (name1==name2)

What will get compared?

A. references will be compared.

B. the two strings will be compared.

C. objects will be compared.

D. error.

9. Consider the above code:

```
String s1=”RAJ”;
String s2=”RAJ”;
 if(s1==s2)
System.out.println(“Hello”);
 else
System.out.println(“Bye”);
```

What will be the output?

A. Bye.

B. Hello.

C. Exception.

D. None of the above.

10. String s1= new String(“RAJ”);

String s2= new String(“RAJ”);

if(s1==s2)

System.out.println(“Hello”);

else

System.out.println(“Bye”);

What will be the output?

A. Bye.

B. Hello.

C. Exception.

D. None of the above.

Subjective Type Question :

1. Write a program to compare two strings entered by the user.

2. Write a program to accept a string from the user and change its case to upper.

3. Write a program which accepts three strings from the user and store them into an array. Now sort them.

4. Write a program to search a particular character in user accepted string and return its position.

5. Write a program to accept two strings from the user and concatenate the two and display them in another string variable without using the inbuilt methods.

6. Write a program to accept a number from the user and convert it into the string without using the inbuilt methods.

7. Write a program to accept a string from the user. Now interchange the first and last character of the string.

8. Write a program to accept a string from the user. Now remove a particular word from that string and append it with a new word.

9. Write a program to accept a string and replace the spaces with " , " character.

10. Write a program to accept a string from the user. Now count the length of string in the terms of each letter. E.g. Sarah is accepted as a string from the user. So the length will be calculated as S-1, A-2, R-3, A-4, H-5.