UNIT I INTRODUCTION TO OOP AND JAVA FUNDAMENTALS

Object Oriented Programming - Abstraction – objects and classes - Encapsulation- Inheritance - Polymorphism- OOP in Java – Characteristics of Java – The Java Environment **-** Java Source File -Structure – Compilation. Fundamental Programming Structures in Java – Defining classes in Java – constructors, methods -access specifiers - static members -Comments, Data Types, Variables, Operators, Control Flow, Arrays , Packages - JavaDoc comments.

1.1 OOPS (OBJECT ORIENTED PROGRAMMING SYSTEM)

In Object oriented programming approach there is a collection of objects. Each object consists of corresponding data structures and the required operations. This is basically the ***bottom up problem*** solving approach**.**

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

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

1.1.1 Difference between Procedural Programming and Object Oriented Programming

Sl. No	Procedural Programming Language	Object Oriented Programming Language
1	Emphasis on algorithm rather than data	Emphasis on data rather than algorithm
2	This is a top down programming approach	This is a bottom up programming approach
3	The major focus is on procedures or functions.	The main focus is on objects
4	Data reusability is not possible	It supports inheritance which provides software reusability
5	Data hiding is not possible	Data hiding can be done by making it private
6	It is simple to implement	It is complex to implement
7	Data allowed to flow freely around the system.	Data are not allowed to flow freely around the system.
8	It has reduced the data security and integrity since the data is available to all functions	It offers high data security and integrity
9	Programmers faced difficulty in design, maintenance and enhancement of software.	It supports polymorphism which reduces program complexity
10	For Example: C. Fortran, COBOL	For Example : C++, JAVA

1.1.2 BASIC CONCEPT OF OOP or CHARACTERISTICS OOPS

1.1.2.1 ABSTRACTION

Act of representing essential features without including background details. Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing. In java, we use abstract class and interface to achieve abstraction.

1.1.2.2 OBJECT

Object is an instance of class. Objects are basic run time entities in object oriented programming. Any entity that has state and behaviour is known as an object. For example: chair, pen, table, keyboard, bike etc. A single class can create any number of objects.

Syntax: `ClassName object = new ClassName();`

1.1.2.3 CLASS

Collection of objects is called class. A class can be defined as an entity in which data and functions are put together.

Syntax:

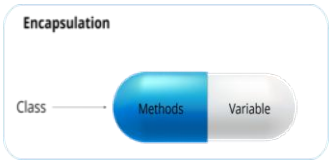
```
Class nameofclass
{
Access specifiers :
      Data member;
      Member function;
}
```

1.1.2.4 ENCAPSULATION

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

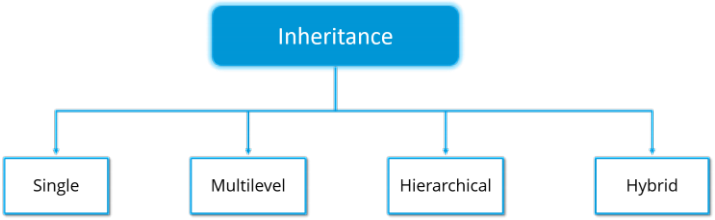
We can achieve encapsulation in Java by:

- Declaring the variables of a class as private.
- Providing public setter and getter methods to modify and view the variables values.



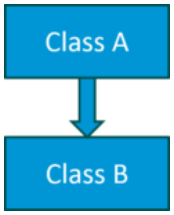
1.1.2.5 INHERITANCE

The properties of base class will be reused in derived class. It provides code reusability. It is used to achieve runtime polymorphism. Inheritance is further classified into 4 types:



Single Inheritance:

In single inheritance, one class inherits the properties of another.



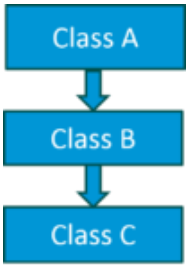
Let's see the syntax for single inheritance:

```
Class A
{
---
}
Class B extends A {
---
}
```

Multilevel Inheritance

When a class is derived from a class which is also derived from another class, i.e. a class having more than one parent class but at different levels, such type of inheritance is called Multilevel Inheritance.

Let's see the syntax for multilevel inheritance in Java:

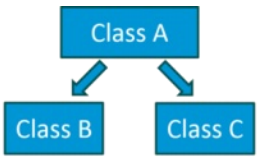



```
Class A{
---
}
Class B extends A{
---
}
Class C extends B{
---
}
```

Hierarchical Inheritance

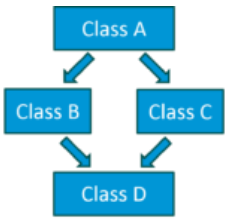
When a class has more than one child classes (sub classes) or in other words, more than one child classes have the same parent class, then such kind of inheritance is known as **hierarchical**.

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



Hybrid Inheritance

Hybrid inheritance is a combination of *multiple* inheritance and *multilevel* inheritance. Since multiple inheritance is not supported in Java as it leads to ambiguity, so this type of inheritance can only be achieved through the use of the interfaces.



1.1.2.6 POLYMORPHISM

Polymorphism means taking many forms, where 'poly' means many and 'morph' means forms. It is the ability of a variable, function or object to take more than one forms.

When *one task is performed by different ways* i.e. known as polymorphism. In java, we use method overloading and method overriding to achieve polymorphism.

Polymorphism in Java is of two types:

1. **Run time polymorphism** - ***Method overriding*** is an example of run time polymorphism.

```
public Class BowlerClass{
void bowlingMethod()
{
System.out.println(" bowler ");
}
public Class FastPacer extends BowlerClass {
void bowlingMethod()
{
System.out.println(" fast bowler ");
}
public static void main(String[] args)
{
FastPacer obj= new FastPacer();
obj.bowlingMethod();
}
}
```

2. **Compile time polymorphism** - ***Method overloading*** is an example of compile time polymorphism. Method Overloading is a feature that allows a class to have two or more methods having the same name but the arguments passed to the methods are different.

```
class Adder {
Static int add(int a, int b)
{
return a+b;
}
static double add( double a, double b)
{
return a+b;
}

public static void main(String args[])
{
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));
}
}
```

Dynamic Method Dispatch

The dynamic method dispatch is also called as ***runtime polymorphism***. During the run time polymorphism, a call to overridden method is resolved at run time.

```
class Base
{
void display()
{
System.out.println("\n Base Method Called");
}
}
class Derived extends Base
{
void display()
{
System.out.println("\n Derived Method Called");
}
}
public class Demo
{
public static void main(String args[])
{
Base obj=new Derived();
obj.display();   //Method invocation determined at run time
}
}
```

Output:

Derived Method Called

1.2 CHARACTERISTICS OF JAVA or FEATURES OF JAVA

Java can be complied and interpreted

Normally programming language can be either compiled or interpreted but Java is a language which can be compiled as well as interpreted. First, Java compiler translates the Java source program into a special code called **bytecode**. Then Java interpreter interprets this bytecode to obtain the equivalent machine code. This machine code is then directly executed to obtain the output.

Java is a Platform Independent (Why java has been called as "Write Once and Run Anywhere"? Explain)

Platform independence means java program can be executed on variety of platforms, such as UNIX, Windows, Linux, Mac, etc. This feature is makes java into – *write once run anywhere*. Because of JVM.

Java is a Portable programming language

Java achieves portability with the help of *bytecode*. Java compiler generates bytecode which can be executed on any platform which have Java Virtual Machine. Primitive data types used in Java are machine independent.

Java is known as Object Oriented Programming Language

Everything in is an object. In java all the code and data lies within the classes and object.

Java is Robust and Secure

Robust simply means strong. Java uses strong memory management. There is lack of pointers that avoids security problem. There is automatic garbage collection in java. There is exception handling and type checking mechanism in java. All these points makes java robust.

Java is designed for distributed system

This feature is very much useful in networking environment. In Java, two different object on different computers can communicate with each other. This can be achieved by Remote Method Invocation (RMI).

Java is small and simple programming language

Java is very simple programming language. Even though you have no programming background you can learn this language very easily. The programmers who have worked on C++ can learn this language very efficiently.

Java is a multithreaded and interactive language

Using Multithread programming features programmer can perform many tasks simultaneously. This allows the programmer to develop the interactive systems.

Java is known for its high performance, scalability monitoring and manageability

Due to the use of bytecode the Java has high performance. The use of multi-threading also helps to improve the performance of the Java. The J2SE helps to increase the scalability in Java.

For monitoring and management Java has large number of Application Programming Interfaces (API).

Java is a dynamic language

This language is capable of dynamically linking new class libraries, methods and objects. Java also supports the functions written in C and C++. These functions are called native methods.

Architectural Neutral

There is no implementation dependent features e.g. size of primitive types is fixed. In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. But in java, it occupies 4 bytes of memory for both 32 and 64 bit architectures.

1.3 THE JAVA ENVIRONMENT

- The Java environment is made up of three things- *development tools, classes and methods* and *java runtime environment*.
- The java development tools constitute ***Java Development Kit (JDK)***
- The classes and methods are called ***Application Programming Interface (API)*** which forms a library called Java Standard Library (JSL)
- The java runtime environment is supported by ***Java Virtual Machine (JVM)***

1.3.1 Java Development Kit (JDK)

- The JDK is nothing but a collection of tools that are used for development and runtime programs.
- It consists of
  - **javac –** The Java compiler which translates the source code to the bytecode form and stores it in a separate class file.
  - **java –** The java interpreter, which interprets the bytecode stored in the class file and executes the program to generate output.
  - **javadoc –** For creating the HTML document for documentation from source code file.
  - **jdb –** The java debugger which helps to find the errors in the program.
  - **appletviewer –** For executing the java applet.
- Following are the steps that illustrate execution process of the application program
  - The user creates the Java source code in the text editor
  - The source code is compiled using the **javac** command
  - The **javadoc** tool can be used to create the HTML files that document the source program.
  - On compiling the source program a class file gets generated which consists of the byte code.
  - The byte code produced by **javac** can be interpreted using **java** in order to produce an executable.

1.3.2 Java Development Kit

The Java API consists of large number of classes and methods. These classes and methods are grouped into package. Examples of some commonly used java packages are-

- Input & output package
- Language support package
- Networking package
- Applet package

1.3.3 Java Development Kit

The Java Runtime Environment (JRE) supports for execution of java programs. It consists of following components.

- **Java Virtual Machine (JVM) :** The JVM takes the byte as an input, reads it, interprets it and then executes it. The JVM can generate output corresponding to the underlying platforms (operating system). Due to JVM the Java programs become portable.
- **Run Time Class Libraries:** These libraries consist of core class libraries that are required for the execution of Java program.
- **Graphical User InterfaceToolkit:** In Java there are two types of GUI toolkits. AWT & swing.
- **Deployment Technologies:** The java-plugins are available that helps in the execution of Java applet on the web browser.

1.4 STRUCTURES JAVA OF JAVA PROGRAM

A Java program involves the following sections:

Documentation Section
Package Statement
Import Statements
Interface Statement
Class Definition

Main method class
{
Public static void main(String args[])
{
//Main method definition
}
}

- ***Documentation Section*** You can write a comment in this section. Comments are beneficial for the programmer because they help them understand the code. These are optional, but we suggest you use them because they are useful to understand the operation of the program, so you must write comments within the program.
- ***Package statement*** You can create a package with any name. A package is a group of classes that are defined by a name. That is, if you want to declare many classes within one element, then you can declare it within a package. It is an optional part of the program, i.e., if you do not want to declare any package, then there will be no problem with it, and you will not get any errors. Here, the package is a keyword that tells the compiler that package has been created. It is declared as: `package package_name;`

  Import statements This line indicates that if you want to use a class of another package, then you can do this by importing it directly into your program. Eg: `import calc.add;`
- ***Interface statement*** Interfaces are like a class that includes a group of method declarations. It's an optional section and can be used when programmers want to implement multiple inheritances within a program.
- ***Class Definition*** A Java program may contain several class definitions. Classes are the main and essential elements of any Java program.
- ***Main Method Class*** Every Java stand-alone program requires the main method as the starting point of the program. This is an essential part of a Java program. There may be many classes in a Java program, and **only one class defines the main method**. Methods contain data type declaration and executable statements.

Here is an example of the Hello Java program to understand the class structure and features. There are a few lines in the program, and the primary task of the program is to print ***Hello Java*** text on the screen.

```
//Name of this file will be "Hello.java"
public class Hello
{
    /* Date: 2018-04-28
       Description:Writes the words "Hello Java" on the screen */
    public static void main(String[] args)
    {
        System.out.println("Hello Java");
    }
}
```

Output:
```
Program Output
Hello Java
```

Here are the most important points to note about the Java programs:

- You have to keep in mind that, Java code is case sensitive.
- To write Java program, you must have to define class first.
- The name of the class in Java (which holds the main method) is the name of the Java program, and the same name will be given in the filename. As mentioned above in the sample program; The name of the class is "Hello" in which the main method is, then this file will be named "Hello.Java".

Let's look into Various Parts of the Above Java Program

public class Hello	• This creates a class called Hello. • All class names must start with a capital letter. • The public word means that it is accessible from any other classes.
/* Comments */	The compiler ignores comment block. Comment can be used anywhere in the program to add info about program or code block, which will be helpful for developers to understand the existing code in the future easily.
Braces	Two curly brackets ***{...}*** are used to group all the commands, so it is known that the commands belong to that class or method.
public static void main	• When the main method is declared public, it means that it can also be used by code outside of its class, due to which the main method is declared public. • The word static used when we want to access a method without creating its object, as we call the main method, before creating any class objects. • The word void indicates that a method does not return a value. main() is declared as void because it does not return a value. • ***main*** is a method; this is a starting point of a Java program.
String[] args	It is an array where each element of it is a string, which has been named as "args". If your Java program is run through the console, you can pass input parameter, and main() method takes it as input.
System.out.println();	This statement is used to print text on the screen as output, where ***system*** is a predefined class, and ***out*** is an object of the PrintWriter class defined in the system. The method ***println*** prints the text on the screen with a new line. You can also use ***print()*** method instead of println() method. All Java statement ends with a semicolon.

1.5 LITERALS

Literal: Any constant value which can be assigned to the variable is called as literal/constant.

```
// Here 100 is a constant/literal.
int x = 100;
```

1.5.1 Integral literals

For Integral data types (byte, short, int, long), we can specify literals in 4 ways:-

1. **Decimal literals (Base 10) :** In this form the allowed digits are 0-9.
   ```
   int x = 101;
   ```
2. **Octal literals (Base 8) :** In this form the allowed digits are 0-7.
   ```
   // The octal number should be prefix with 0.
   int x = 0146;
   ```
3. **Hexa-decimal literals (Base 16) :** In this form the allowed digits are 0-9 and characters are a-f. We can use both uppercase and lowercase characters. As we know that java is a case-sensitive programming language but here java is not case-sensitive.
   ```
   // The hexa-decimal number should be prefix
   // with 0X or 0x.
   int x = 0X123Face;
   ```
4. **Binary literals :** From 1.7 onward we can specify literals value even in binary form also, allowed digits are 0 and 1. Literals value should be prefixed with 0b or 0B.
   ```
   int x = 0b1111;
   ```

NOTE : By default, every literal is of int type, we can specify explicitly as long type by suffixed with l or L. There is no way to specify byte and short literals explicitly but indirectly we can specify. Whenever we are assigning integral literal to the byte variable and if the value within the range of byte then the compiler treats it automatically as byte literals.

Example

```
// Java program to illustrate the application of Integer literals
public class Test {
    public static void main(String[] args)
    {
        int a = 101; // decimal-form literal
        int b = 0100; // octal-form literal
        int c = 0xFace; // Hexa-decimal form literal
        int d = 0b1111; // Binary literal
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println(d);
    }
}
```

Output

```
101
64
64206
15
```

1.5.2 Floating-Point literal

For Floating-point data types, we can specify literals in only decimal form and we cant specify in octal and Hexa-decimal forms.

Decimal literals(Base 10) : In this form the allowed digits are 0-9.

```
double d = 123.456;
```

```
// Java program to illustrate the application of
// floating-point literals
public class Test {
    public static void main(String[] args)
    {
        float a = 101.230; // decimal-form literal
        float b = 0123.222; // It also acts as decimal literal
        float c = 0x123.222; // Hexa-decimal form
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
}
```

Output

```
101.230
123.222
Error: malformed floating point literal
```

NOTE: By default every floating point literal is of double type and hence we cant assign directly to float variable. But we can specify floating point literal as float type by suffixed with f or F. We can specify explicitly floating point literal as double type by suffixed with d or D. Of course this convention is not required.

1.5.3 Char literal

For char data types we can specify literals in 4 ways:

1. **Single quote :** We can specify literal to char data type as single character within single quote.
   ```
   char ch = 'a';
   ```
2. **Char literal as Integral literal :** we can specify char literal as integral literal which represents Unicode value of the character and that integral literals can be specified either in Decimal, Octal and Hexadecimal forms. But the allowed range is 0 to 65535.
   ```
   char ch = 062;
   ```
3. **Unicode Representation :** We can specify char literals in Unicode representation '\uxxxx'. Here xxxx represents 4 hexadecimal numbers.
   ```
   char ch = '\u0061';// Here /u0061 represent a.
   ```
4. **Escape Sequence :** Every escape character can be specify as char literals.
   ```
   char ch = '\n';
   ```

```
// Java program to illustrate the application of char literals
public class Test {
public static void main(String[] args)
    {
        char ch = 'a'; // signle character literl within signle quote
        char b = 0789; // It is an Integer literal with octal form
        char c = '\u0061'; // Unicode representation
        System.out.println(ch);
        System.out.println(b);
        System.out.println(c);
        // Escape character literal
        System.out.println("\"  is a symbol");
    }
}
```

Output

```
a
error:Integer number too large
a
"  is a symbol
```

1.5.4 String literal

Any sequence of characters within double quotes is treated as String literals.

```
String s = "Hello";
```

String literals may not contain unescaped newline or linefeed characters. However, the Java compiler will evaluate compile time expressions, so the following String expression results in a string with three lines of text:

Example:

```
String text = "This is a String literal\n"
            + "which spans not one and not two\n"
            + "but three lines of text.\n";
// Java program to illustrate the application of String literals
public class Test {
    public static void main(String[] args)
    {
        String s = "Hello";

        // If we assign without "" then it treats as a variable
        // and causes compiler error
        String s1 = Hello;

        System.out.println(s);

        System.out.println(s1);
    }
}
```

Output

```
Hello
error: cannot find symbol
symbol:   variable Hello
location: class Test
```

1.5.5 boolean literals

Only two values are allowed for Boolean literals i.e. true and false.

```
boolean b = true;
// Java program to illustrate the application of boolean
// literals
public class Test {
    public static void main(String[] args)
    {
        boolean b = true;
        boolean c = false;
        boolean d = 0;
        boolean b = 1;
        System.out.println(b);
        System.out.println(c);
        System.out.println(d);
        System.out.println(e);
    }
}
true
false
error: incompatible types: int cannot be converted to boolean
error: incompatible types: int cannot be converted to boolean
```

NOTE : When we are performing concatenation operations, then the values in brackets are concatenated first. Then the values are concatenated from the left to the right. We should be careful when we are mixing character literals and integers in String concatenation operations and this type of operations are known as **Mixed Mode operation**.

1.6 DEFINING CLASSES IN JAVA

A class is a collection of data members and methods that operates on the data. The class must be declared either default or public.

Syntax:

```
Access-Modifier  class classname
{
Acces_Modifier datatype vriablename-1;
Acces_Modifier datatype vriablename-2;
  modifier returnType nameOfMethod (Parameter List)
  {
     // method body
  }
}
```

Access Modifier

The access modifiers in java specifies scope of a data member, method, constructor or class. There are 4 types of java access modifiers:

- **Private** - The private access modifier is accessible only within class.
- **Default** - If you don't use any modifier, it is treated as default bydefault. The default modifier is accessible only within package.
- **Protected** - The protected access modifier is accessible within package and outside the package but through inheritance only.
- **Public** - The public access modifier is accessible everywhere. It has the widest scope among all other modifiers.

Example:

```
public class Demo
{
public int a,b,c;                //Instance Variable
public void sum(int x, int y)    //Instance Method
{
a=x;
b=y;
c=a+b;
System.out.println(c);
}
}
class MainDemo
{
public static void main(String args[])
{
Demo obj = new Demo();      //Object Declaration
obj.sum(10,20);            //Accessing class member using object
}
}
```

1.6.1 OBJECT

Objects are nothing but instance of class. By using objects we can able to access the class members.

Syntax:

```
Classname object = new classname();
```

Example:

```
Demo obj = new Demo();
```

Accessing the Class Members

There are two types of class members such as data members and method. Class members can be accessed using **dot** operator.

Syntax:

```
Name_of_object.Variable_name=value;
Name_of_objecy.methodname(parameter list);
```

Example:

```
Obj.a=10;   // Accessing variable
Obj.b=20;
Obj.sum(a,b);
```

1.7 CONSTRUCTORS

- **Constructor** is a **special type of method** that is used to initialize the object.
- Constructor is executed **at the time of object creation**.
- Constructor name must be same as its class name.
- Constructor may be defined with or without arguments.
- It does not have any return type.

Example of parameterized constructor

```
class Circle
{

float radius,aoc;

Circle()//default constructor
{
radius = 0;
aoc = 0;
}

Circle(float r) //parameterized constructor
{
radius = r;
}

float getArea()
{
aoc=3.14 * radius * radius;
}

void display()
{
System.out.println("Area of Circle is:" + aoc);
}

public static void main(String args[])
{
Circle c1 = new Circle();
Circle c2 = new Circle(2.5);
c1.getArea();
c2.getArea();
c1.display();
c2.display();
}
}
```

```
class add
{
int a,b,c;
add()  //default constructor
{
a=0;
b=0;
c=0;
}
add(int x, int y) //parameterized constructor
{
a=x;
b=y;
c=0;
}
add(add obj) //Copy constructor
{
a=obj.a;
b=obj.b;
c=obj.c;
}
void calculate()
{
c=a+b;
System.out.println(c);
}

public static void main(String args[])
{
add obj = new add()
add obj1 = new add(10,20);
add obj2 = new add(obj1);
obj.calculate();
obj1.calculate();
obj2.calculate();
}
}
```

1.8 METHODS

With an example, discuss how to declare methods with multiple parameters in Java. (8) (N/D 17)

Method is a collection of statements that are grouped together to perform an operation.

Syntax

```
modifier returnType nameOfMethod (Parameter List) {
   // method body
}
```

Example :

```
public class Demo{
public static void main(String args[]){
public int a=10,b=20,c;
sum(a,b);                        //Method Calling
}
public void sum(int x, int y)   // Method Definition
{
a=x;
b=y;
c=a+b;
System.out.println(c);
}
}
```

1.8.1 Method Overriding

Rules for method overriding

- The return type, function name, and argument list must be same in both super and sub class. But implementation may be different.
- If the superclass method is declared public then the overriding method in the sub class cannot be either private or protected.
- Instance methods can be overridden only if they are inherited by the subclass.
- A method declared final cannot be overridden.
- A method declared static cannot be overridden but can be re-declared.
- If a method cannot be inherited, then it cannot be overridden.
- Constructors cannot be overridden.

Example

```
class A {
      int regno=1030;
      void print()
      {
      System.out.println("reno"+regno);
      }
}
class B extends A
{
      string name="latha";
      void print()
      {
      Super.Print();
      System.out.println("name"+name);
      }
}
```

```
class Demo
{
      public static void main (String[] args)
      {
      B obj2 = new B();
      obj2.print();
      }
}
```

OUTPUT

```
C:\Java\jdk1.5\bin>javac student.java
C:\Java\jdk1.5\bin>java student
Reno 1030
Name latha
```

1.8.2 Method Overloading

If a class has multiple methods having same name but different in parameters, it is known as Method Overloading.

```
class Adder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
class TestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
}
}
```

FINALIZE METHOD

Finalize() is the method of Object class. This method is called just before an object is garbage collected. finalize() method overrides to dispose system resources, perform clean-up activities and minimize memory leaks.

Syntax

protected void finalize() throws Throwable

Throwable - the Exception is raised by this method

Example

```
public class JavafinalizeExample1 {
    public static void main(String[] args)
    {
       JavafinalizeExample1 obj = new JavafinalizeExample1();
       obj = null;
       // calling garbage collector
       System.gc();
       System.out.println("end of garbage collection");
    }
    protected void finalize()
    {
       System.out.println("finalize method called");
    }
}
```

Output:

```
end of garbage collection
finalize method called
```

1.9 ACCESS MODIFIER

What is an access modifiers? Differentiate between private, protected and public access modifiers with examples. (16) (A/M 15)

Java Access Specifiers (also known as Visibility **Specifiers**) regulate **access** to classes, fields and methods in **Java**.These **Specifiers** determine whether a field or method in a class, can be used or invoked by another method in another class or sub-class. **Access Specifiers** can be used to restrict **access**.

The access modifiers in java specify scope of a data member, method, constructor or class. There are 4 types of java access modifiers:

- ***Private*** - The private access modifier is accessible only within class.
- ***Default*** - If you don't use any modifier, it is treated as default bydefault. The default modifier is accessible only within package.
- ***Protected*** - The protected access modifier is accessible within package and outside the package but through inheritance only.
- ***Public*** - The public access modifier is accessible everywhere. It has the widest scope among all other modifiers.

1.9.1 Private access modifier

```
class A{
private int data=40;
private void msg()
{
System.out.println("Hello java");
}
}
public class Demo{
public static void main(String args[]){
A obj=new A();
System.out.println(obj.data);    //Compile Time Error
obj.msg();                       //Compile Time Error
}
}
```

In this example, we have created two classes A and Demo. A class contains private data member and private method. We are accessing these private members from outside the class, so there is compile time error.

1.9.2 Default access modifier

```
//save by A.java
package pack;
class A{
void msg(){System.out.println("Hello");}
}
//save by B.java
package mypack;
import pack.*;
class B{
```

```
public static void main(String args[]){
A obj = new A();       //Compile Time Error
obj.msg();             //Compile Time Error
}  }
```

In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

1.9.3 Public access modifier

```
//save by A.java
    package pack;
    public class A{
    public void msg(){System.out.println("Hello");}
    }
//save by B.java
    package mypack;
    import pack.*;
    class B{
      public static void main(String args[]){
       A obj = new A();
       obj.msg();
      }  }          Output:Hello
```

1.9.4 Protected access modifier

```
//save by A.java
   package pack;
   public class A{
   protected void msg(){System.out.println("Hello");}
   }
//save by B.java
   package mypack;
   import pack.*;
     class B extends A{
     public static void main(String args[]){
      B obj = new B();
      obj.msg();
     }
   }
```

In above example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

1.10 STATIC MEMBERS

Distinguish between instance method and class method with an example.

Sl. NO	Instance Method	Class Method or Static Method
1	Non Static methods are called as instance method	Static Methods are called as Class Method
2	Instance method can access the instance methods and instance variables directly.	Static methods can access the static variables and static methods directly.
3	Instance method can access static variables and static methods directly.	Static methods **can't** access instance methods and instance variables directly.
4	Instance members are not shared by all objects.	Static members are shared by all the objects

5	Class Test
	{
	public int a=0;
	public void count()
	{
	a++;
	System.out.println(a);
	}
	}
	
	Class Demo
	{
	public static void main(String args[])
	{
	Test obj1 = new Test();
	obj1.count();
	Test obj2 = new Test();
	obj2.count();
	}
	}
		Class Test
		{
		public static int a=0;
		public static void count()
		{
		a++;
		System.out.println(a);
		}
		}
		
		Class Demo
		{
		public static void main(String args[])
		{
		Test.count();
		Test obj1 = new Test();
		obj1.count();
		}
		}

6	Output: 1 1	Output: 1 2

1.11 DATA TYPES IN JAVA

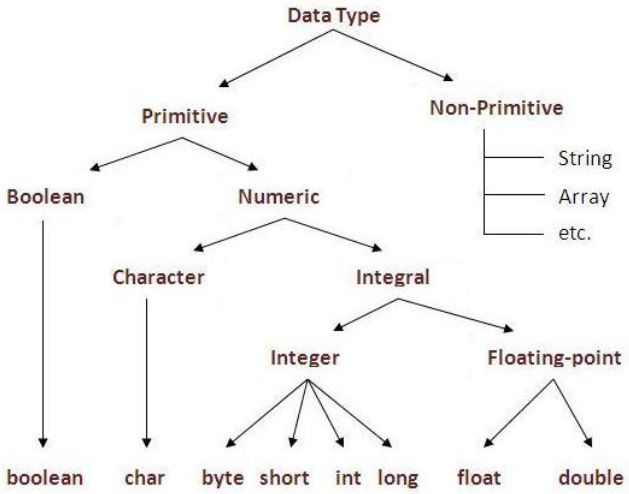
Data types represent the different values to be stored in the variable. In java, there are two types of data types:

- Primitive data types
- Non-primitive data types

Data Type	Size	Range	Example
byte	1 byte	-128 to 127	byte a = 100 , byte b = -50
short	2 byte	-32,768 to 32,767	short a = 10000, short r = -20000
int	4 byte	2,147,483,648 to 2,147,483,647	int a = 100000, int b = -200000
long	8 byte	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	long a = 100000L, int b = -200000L
float	4 byte	1.4e–045 to 3.4e+038	float a = 234.5f
double	8 byte	4.9e–324 to 1.8e+308	double a = 123.4
char	2 byte	0 to65,536	char a ='A'
boolean	1 bit	0 or 1	boolean a = true;

Example

```
class Demo
{
public static void main(String args[])
{
byte a=10;
short b=10*128;
int c=10000*128;
long d=10000*1000*128;
double e=99.9998;
char f=’a’;
boolean g=true;
boolean h=false;
System.out.println(“The value of a=:”+a);
System.out.println(“The value of b=:”+b);
System.out.println(“The value of c=:”+c);
System.out.println(“The value of d=:”+d);
System.out.println(“The value of e=:”+e);
System.out.println(“The value of f=:”+f);
f++;
System.out.println(“The value of f after increment=:”+f);
System.out.println(“The value of g=:”+g);
System.out.println(“The value of h=:”+h);
} }
```



Output:

The value of a= 10
The value of b= 1280
The value of c= 1280000
The value of d= 1280000000
The value of e= 99.9998
The value of f= a
The value of f after increment= b
The value of g= true
The value of h=false

1.11.1 Type Casting

Assigning a value of one type to a variable of another type is known as Type Casting.

Example :

```
int x = 10;
byte y = (byte)x;
```

In Java, type casting is classified into two types,

1. Widening Casting(Implicit) or Automatic type conversion
2. Narrowing Casting(Explicitly done)

Widening Casting(Implicit) or Automatic type conversion

- the two types are compatible
- the target type is larger than the source type

Example:

```
public class Test
{
    public static void main(String[] args)
    {
      int i = 100;
      long l = i;       //no explicit type casting required
      float f = l;      //no explicit type casting required
      System.out.println("Int value "+i);
      System.out.println("Long value "+l);
      System.out.println("Float value "+f);
    }

}
```

Output
Int value 100
Long value 100
Float value 100.0

Narrowing Casting(Explicitly done)

When you are assigning a larger type value to a variable of smaller type, then you need to perform explicit type casting.

Example:

```
public class Test
{
    public static void main(String[] args)
    {
      double d = 100.04;
      long l = (long)d;  //explicit type casting required
      int i = (int)l;  //explicit type casting required

      System.out.println("Double value "+d);
      System.out.println("Long value "+l);
      System.out.println("Int value "+i);
    }
}
```

Output
Double value 100.04
Long value 100
Int value 100

1.12 VARIABLES IN JAVA

Variable is a name of memory location. It is an identifier that denotes the storage location. There are three types of variables in java: **local, instance and static.**

1. *Local Variable:* A variable which is declared inside the method is called local variable.
2. *Instance Variable:* A variable which is declared inside the class but outside the method, is called instance variable. It is not declared as static.
3. *Static variable:* A variable that is declared as static is called static variable. It cannot be local.

Following are some rules for variable declaration

- The variable name should not with digits.
- No special character is allowed in identifier except underscore.
- There should not be any blank space with the identifier name.
- The identifier name should not be a keyword.
- The identifier name should be meaningful.

Example to understand the types of variables in java

```
class A{
    int data=50;//instance variable
    static int m=100;//static variable
    void method(){
    int n=90;//local variable
    }
    }//end of class
```

1.13 OPERATORS

Discuss the various types of operators in Java and explain with suitable examples. (16) (A/M 15)

Type	Operator	Meaning	Example
Arithmetic	+	Addition or unary plus	c=a+b
	-	Subtraction or unary minus	c=a-b or c=-a
	*	Multiplication	c=a*b
	/	Division	c=a/b
	%	Mod	c=a%b
Relational	<	Less than	a<4
	>	Greater than	b>10
	<=	Less than equal to	b<=5
	>=	Greater than equal to	a>=5
	==	Equal to	x==5
	!-	Not equal to	m!=9
Logical	&&	And operator	0&&1
	||	Or operator	0||1
Assignment	=	Is assigned to	A=5
Increment	++	Increment by one	++i or i++
Decrement	--	Decrement by one	--k or k--
Conditional	Syntax: Condition?expression1:expresion2 Example: a>b?true:false		

Program-1 Arithmetic Operation

```
class Demo
{
public static void main(String args[])
{
//Arithmetic Operation
int a=10,b=20,c;
c=a+b;
System.out.println("Addition of two number is" +c);
c=a-b;
System.out.println("Subtraction of two number is" +c);
//Relational Operation
System.out.println("The a<b is" +(a<b));
System.out.println("The a>b is" +(a>b));
System.out.println("The a==b is" +(a==b));
//Increment and Decrement operator
System.out.println("Increment of a is " + a++);
System.out.println("Decrement of a is " + a--);
} }
```

Output:

Addition of two number is 30

Subtraction of two number is -10

The a<b is True

The a>b is False

The ==b is False

Increment of a is 11

Decrement of a is 10

1.14 CONTROL STATEMENTS

Programmers can take decisions in their program with the help of control statements, various control statements that can be used in java are

- Conditional Statement
  - If statement
  - If else statement
  - If else if ladder
- Looping statement
  - While statement
  - Do while statement
  - For statement
- Selection statement
  - Switch case statement

1.14.1 Conditional Statement

i) Simple If statement

Syntax:

```
if(condition)
{
//code to be executed
}
```

Example:

```
public class Demo {
public static void main(String[] args) {
int age=20;
if(age>18)
{
System.out.print("Age is greater than 18");
}  }  }
```

ii) If else statement

Syntax:

```
if(condition)
{
//code if condition is true
}
else
{
//code if condition is false
}
```

Example:

```
public class Demo {
public static void main(String[] args) {
     int number=13;
     if(number%2==0){
         System.out.println("even number");
     }else{
         System.out.println("odd number");
     }  } }
```

iii)If else if ladder statement

Syntax:

```
if(condition1){
  //code to be executed if condition1 is true
  }else if(condition2){
  //code to be executed if condition2 is true
  }
  else if(condition3){
  //code to be executed if condition3 is true
  }
  ...
  else{
  //code to be executed if all the conditions are false
  }
```

Example:

```
public class Demo {
public static void main(String[] args) {
    int marks=65;
    if(marks<50){
        System.out.println("fail");
    }
    else if(marks>=50 && marks<60){
        System.out.println("D grade");
    }
    else if(marks>=60 && marks<70){
        System.out.println("C grade");
    }
    else if(marks>=70 && marks<80){
        System.out.println("B grade");
    }
    else if(marks>=80 && marks<90){
        System.out.println("A grade");
    }else if(marks>=90 && marks<100){
        System.out.println("A+ grade");
    }else{
        System.out.println("Invalid!");
    }
}
}
```

1.14.2 Looping Statement

i) While statement

While loop will be executed again and again until condition became false. In while loop first check the condition then execute the statement.

Syntax

```
while(condition){
//code to be executed
}
```

Example:

```
public class Demo {
public static void main(String[] args) {
    int i=1;
    while(i<=5){
        System.out.println(i);
        i++;
    }
  }
}
```

ii) do While statement

Do while loop will be executed again and again until condition became false. In do while loop first execute the condition then check the statement.

Syntax

```
do{
//code to be executed
} while(condition);
```

Example:

```
public class Demo {
public static void main(String[] args) {
    int i=1;
    do{
        System.out.println(i);
        i++;
    } while(i<=5);
}
}
```

iii) For loop statement

Syntax:

```
for(initialization;condition;incr/decr){
//code to be executed
}
```

Example-1

```
public class Demo {
public static void main(String[] args) {
    for(int i=1;i<=10;i++){
        System.out.println(i);
    }
}
}
```

1.14.3 Selection Statement

Switch Case Statement

The Java *switch statement* executes one statement from multiple conditions. It is like if-else-if ladder statement.

Syntax:

```
switch(expression){
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
......

default:
 code to be executed if all cases are not matched;
}
```

Example:

```
public class SwitchExample {
public static void main(String[] args) {
    int number=20;
    switch(number){
    case 10:
         System.out.println("10");
         break;
    case 20:
         System.out.println("20");
         break;
    case 30:
         System.out.println("30");
         break;
    default:System.out.println("Not in 10, 20 or 30");
    }
  }
}
```

1.14.4 Program-1 Write a Java program to find the sum of the following series 1-2+3-4+….+n

```
class Demo
{
public static void main(String args[])
{
int i,n=10,sum=0;
for(i=1;i<=n;i=i+2)
{
sum=sum+i;
}
for(i=2;i<=n;i+2)
{
sum=sum-i;
}
System.out.println("\n Sum of series: "+sum);
}
```

Program-2 Write a Java program to find factorial of a given number.

```
import java.io.*;
class Demo {
public static void main(String args[]){
int n=5, fact=1;
for(int i=1;i<=n;i++)
{
fact=fact*i;
}
System.out.println(fact);
}
}
```

Program-3 Write a Java program for following series $1+1/2+1/2^2+1/2^3+…+1/2^n$.

```
import java.io.*;
import java.lang.*;
import java.math.*;
public class Demo
{
public static void main(String args[])throes IOException
{
int n=5;
double val,sum=1;
for(int i=1;i<n;i++)
{
val=1/(Math.pow(2,i));
sum=sum+val;
}
System.out.println("\n Sum of the series is"+sum);
}}
```

Program-4 Let's see the simple example of the Java Scanner class which reads the int, string and double value as an input:

```
import java.util.Scanner;
class Demo{
public static void main(String args[]){
Scanner sc=new Scanner(System.in);
System.out.println("Enter your rollno");
int rollno=sc.nextInt();
System.out.println("Enter your name");
String name=sc.next();
System.out.println("Enter your fee");
double fee=sc.nextDouble();
System.out.println("Rollno:"+rollno+" name:"+name+" fee:"+fee);
sc.close();
}
}
```

```
Enter your rollno
 111
 Enter your name
 Ratan
 Enter your fee
 450000
 Rollno:111 name:Ratan fee:450000
```

Program-5 Write a Java program to find the maximum of the given array.

```
import java.io.*;
class Demo
{
public static void main(String args[])throws IOException
{
int a[]={10,50,20,40,30};
int max;
max=a[0];
for(int i=0;i<5;i++)
{
if(max<a[i])
{
max=a[i];
}
}
System.out.println("\n The maximum number is:"+max);
}}
```

Program-6 Display Fibonacci series using for loop (8) (N/D 17)

```
public class Fibonacci {
    public static void main(String[] args) {
        int n = 10, t1 = 0, t2 = 1;
        System.out.print("First " + n + " terms: ");
        for (int i = 1; i <= n; ++i)
        {
            System.out.print(t1 + " + ");
            int sum = t1 + t2;
            t1 = t2;
            t2 = sum;
        }
    }
}
```

Program-7 Sorting of N Strings/Names in Ascending Order using Java program

```
//Sorting Names
import java.util.Scanner;
class Demo
{
public static void main(String args[])
{
String temp;
Scanner SC = new Scanner(System.in);
System.out.print("Enter the value of N: ");
int N= SC.nextInt();
String names[] = new String[N];

System.out.println("Enter names: ");
for(int i=0; i<N; i++)
{
System.out.print("Enter name:");
names[i] = SC.nextLine();
}
for(int i=0; i<5; i++)
{
for(int j=1; j<5; j++)
{
if(names[j-1].compareTo(names[j])>0)
{
temp=names[j-1];
names[j-1]=names[j];
names[j]=temp;
}
}
}
for(int i=0;i<N;i++)
{
System.out.println(names[i]);
}}}
```

```
//Sorting Numbers
import java.util.Scanner;
public class Demo
{
public static void main(String[] args)
{
int n, temp;
Scanner s = new Scanner(System.in);
System.out.print("Enter the array limit:");
n = s.nextInt();
int a[] = new int[n];
System.out.println("Enter the elements:");
for (int i = 0; i < n; i++)
{
a[i] = s.nextInt();
}
for (int i = 0; i < n; i++)
{
for (int j = i + 1; j < n; j++)
{
if (a[i] > a[j])
{
temp = a[i];
a[i] = a[j];
a[j] = temp;
}
}
}
System.out.print("Ascending Order:");
for (int i = 0; i < n - 1; i++)
{
System.out.print(a[i] + ",");
}
System.out.print(a[n - 1]);
}
}
```

1.15 ARRAYS

Array is a collection of similar data elements that are stored under a contiguous memory location.

Array must be declared and created in the computer memory before they are used. Creation of an array involves three steps,

1. Declaring an array
2. Creating memory location for an array or (array instantiation)
3. Initialization of array or (Putting values into the memory locations or array)

Types of Array in java

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

1. Declaring an array:

1. dataType array-name[ ]; (or)
2. dataType [ ]array-name; (or)

Example:

1. int arr[ ]; (or)
2. int [ ]arr; (or)

2. Creation memory location for an array:

After declaring an array, we need to create memory for an array. Java allows us to create memory for arrays using **new** operator.

Syntax:

array-name [ ] = new dataType[size];

Example:

arr[ ] = new int[5];

(or)

In one Step:

Array declaration and memory allocation can be done in one step,

Syntax:

dataType array-name[ ] = new dataType[size]

Example:

int arr[ ] = new int[5];

3. Initialization of array:

Array can be initialized in two ways either compile time or run time.

Syntax for Compile Time initialization:

dataType array-name = {intital values};

Example:

int arr[ ] = {1,2,3,4,5}

In this method, no need to use new operator to allocate space to the array.

Syntax for Run Time initialization:

dataType array-name = new dataType[size];
array-name[3]=78;

Example:

int arr[ ] = {1,2,3,4,5}
arr[3]=78;

Example program for Compile time initialization:

```
import java.io.*;
class average
{
public static void main (String args[])
{
int num[ ] = {10,11,12,13,14};
int result = 0;
float avg;
for(i=0;i<5;i++)
{
result = result + num[i];
}
avg=result/5;
System.out.println("Average is " + avg);
}
}
```

OUTPUT:

Average is 12

Example program for Run time initialization:

```
import java.io.*;
class average
{
public static void main (String args[])throws IOException
{
DataInputStream din = new DataInputStream(System.in);
int num[ ] = new int[5];
int result = 0;
float avg;
System.out.println("Enter the marks: ");
for(i=0;i<5;i++)
{
num[i]=Integer.parseInt(din.readLine());
}
for(i=0;i<5;i++)
{
result = result + num[i];
}
avg=result/5;
System.out.println("Average is " + avg);
}
}
```

1.15.2 MULTIDIMENSIONAL ARRAYS:

Multidimensional arrays are specified using multiple indexes,

Array Declaration:

int a[ ][ ]; (or) int [ ][ ]a;

Creation memory location for an array:

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

Initialization of array:

```
int a[ ][ ]={{10,20},{30,40}};
```

Program for Compile time initialization

```
import java.io.*;
class Testarray3
{
 public static void main(String args[])
{
 int arr[][]={{1,2,3},{4,5,6},{7,8,9}};   //declaring and initializing 2D array
//printing 2D array
for(int i=0;i<3;i++)
{
for(int j=0;j<3;j++)
{
System.out.print(arr[i][j]+" ");
}
System.out.println();
}}  }
```

Output:

```
1 2 3
4 5 6
7 8 9
```

Example program for Compile time initialization

```
import java.io.*;
class Testarray3
{
 public static void main(String args[])
{
DataInputStream din = new DataInputStream(System.in);
 int arr[][]= new int[3][3];
System.out.println("Enter the matrix elements");
for(int i=0;i<3;i++)
{
for(int j=0;j<3;j++)
{
arr[i][j]= Integer.parseInt(din.readLine());
}
System.out.print("The transpose of matrix is:");
for(int i=0;i<3;i++)
{
for(int j=0;j<3;j++)
{
System.out.print(arr[j][i]+" ");
}
System.out.println();
 }
}
}
```

Output:

```
Enter the matrix elements
1 2 3
4 5 6
7 8 9
The transpose of matrix is
1 4 7
2 5 8
3 6 9
```

Run time: Example program for addition of matrix

```
import java.io.*;
class Testarray3
{
 public static void main(String args[])
{
DataInputStream din = new DataInputStream(System.in);
int a[][]= new int[3][3];
int b[][]= new int[3][3];
int c[][]= new int[3][3];
System.out.println("Enter the first matrix elements:");
for(int i=0;i<3;i++)
{
for(int j=0;j<3;j++)
{
arr[i][j]= Integer.parseInt(din.readLine());
}
System.out.println("Enter the second matrix elements:");
for(int i=0;i<3;i++)
{
for(int j=0;j<3;j++)
{
arr[i][j]= Integer.parseInt(din.readLine());
}
System.out.print("The sum of two matrix is:");

for(int i=0;i<3;i++)
{
for(int j=0;j<3;j++)
{
c[i][j]=a[i][j]+b[i][j];
System.out.print(c[j][i]+" ");
}
System.out.println();
}} }
```

Output:

```
Enter the first matrix elements
        2 2 2
        2 2 2
        2 2 2
Enter the first matrix elements
        2 2 2
        2 2 2
        2 2 2
The sum of two matrix is
       4 4 4
       4 4 4
       4 4 4
```

Compile Time: Example program for addition of matrix:

```
import java.io.*;
class Testarray5{
 public static void main(String args[]){
//creating two matrices
    int a[][]={{1,3,4},{3,4,5}};
    int b[][]={{1,3,4},{3,4,5}};
 //creating another matrix to store the sum of two matrices
    int c[][]=new int[2][3];
//adding and printing addition of 2 matrices
    for(int i=0;i<2;i++){
    for(int j=0;j<3;j++){
    c[i][j]=a[i][j]+b[i][j];
    System.out.print(c[i][j]+" ");
    }
    System.out.println();//new line
    }
    }}
```

1.16 PACKAGE

Package is a collection of related classes and interfaces. It is also defined as **"**putting classes together"

1.16.1 Types of Packages

- Java API Packages
- User Defined Packages

1.16.2 Java API Packages

- Language Package(java.lang)
- Utilities Package(java.util)
- I/O Package(java.io)
- Networking Package(java.net)
- Applet Package(java.applet)
- AWT Package(java.awt)

1.16.3 User defined packages

To create a package simply includes a package commands as the first statement in the java source file. Any classes declared within that file will belong to specified package.

Creation of Package pack1:

1. Create one directory in the name of pack1.
2. Change directory to pack1.
3. Design the classes belongs to this pack1.
4. Save and Compile the classes individually.

Creation of Package pack2:

1. Create another directory in the name of pack2.
2. Change directory to pack2.
3. Design the classes belongs to this pack2.
4. Save and Compile the classes individually.

5.1.4 Example program for package - 1:

```
package pack1;  // create a class aa belongs to pack1
public class aa
{
     public void display1()
     {
          System.out.println("This is from pack1..");
     }
}                              //Note: Type this program inside the pack1 directory and compile it.
```

```
package pack2;  // create a class bb belongs to pack2
public class bb
{
     public void display2()
     {
          System.out.println("This is from pack2..");
     }
}                         //Note:Type this program inside the pack2 directory and compile it.

package pack1.subpack;      // create a class cc belongs to sub package subpack
public class cc
{
     public void display3()
     {
          System.out.println("This is from sub package of
pack1..");
     }
}
```

Note: Make a new subdirectory 'subpack' for pack1 directory. Type this program inside the subpack directory and compile it.

```
import java.io.*;           // Main program for accessing all newly created packages.
import pack1.*;
import pack2.*;
import pack1.subpack.*;
class PackDemo
{
     public static void main(String args[])
     {
          aa a1=new aa();
          bb b1=new bb();
          cc c1=new cc();
          a1.display1();
          b1.display2();
          c1.display3();
     }
}                    //Note: Compile and Run this program from the parent directory of pack1.
```

OUTPUT:
This is from pack1..
This is from pack2..
This is from sub package of pack1..

1.17 JAVADOC (JAVA DOCUMENTATION COMMENTS)

- Java supports three types of comments:
    1. Single line comment *//statement*
    2. Multiline comment */* statement */*
    3. Documentation comment /** *statement */*
- Javadoc is a tool that generates html documentation from Javadoc comments in the code.
- Javadoc allows you to attach descriptions to classes, constructors, fields, interfaces and methods in the generated html documentation by placing Javadoc comments directly before their declaration statements.

Javadoc Tags

- ***Tags*** are keywords recognized by Javadoc which define the type of information that follows.
- Tags come after the description (separated by a new line).
- Here are some common pre-defined tags:
    - ***@author [author name]*** - identifies author(s) of a class or interface.
    - ***@version [version]*** - version info of a class or interface.
    - ***@param [argument name] [argument description]*** - describes an argument of method or constructor.
    - ***@return [description of return]*** - describes data returned by method (unnecessary for constructors and void methods).
    - ***@exception [exception thrown] [exception description]*** - describes exception thrown by method.
    - ***@throws [exception thrown] [exception description]*** - same as ***@exception***.

Here's an example with tags:

```
/** Description of MyClass
 *
 * @author John Doe
 * @author Jane Doe
 * @version 6.0z Build 9000 Jan 3, 1970.
 */
public class MyClass
{
    /** Description of myIntField */
    public int myIntField;
    /** Description of MyClass()
     *
     * @throws MyException Description of myException   */
    public MyClass() throws myException
    {
          // statements
    }
```

```
    /** Description of myMethod(int a, String b)
     *
     * @param a        Description of a
     * @param b        Description of b
     * @return         Description of c
     */
    public Object myMethod(int a, String b)
    {
          Object c;
          // Statements
          return c;
    }
}
```

Javadoc Compilation

- To generate the html documentation, run Javadoc followed by the list of source files, which the documentation is to be generated for, in the command prompt (i.e. ***Javadoc [files]***).
- Javadoc also provides additional options which can be entered as switches following the Javadoc command (i.e. ***Javadoc [options] [files]***).

- Here are some basic Javadoc options:

  - ***-author*** - generated documentation will include a author section
  - ***-classpath [path]*** - specifies path to search for referenced .class files.
  - ***-classpathlist [path];[path];...;[path]*** - specifies a list locations (separated by ";") to search for referenced .class files.
  - ***-d [path]*** - specifies where generated documentation will be saved.
  - ***-private*** - generated documentation will include private fields and methods (only public and protected ones are included by default).
  - ***-sourcepath [path]*** - specifies path to search for .java files to generate documentation form.
  - ***-sourcepathlist [path];[path];...;[path]*** - specifies a list locations (separated by ";") to search for .java files to generate documentation form.
  - ***-version*** - generated documentation will include a version section

 Some examples

- Basic example that generates and saves documentation to the current directory (c:\MyWork) from A.java and B.java in current directory and all .java files in c:\OtherWork\.
  - *c:\MyWork>* ***Javadoc A.java B.java c:\OtherWork*.java***
- More complex example with the generated documentation showing version information and private members from all .java files in c:\MySource\ and c:\YourSource\ which references files in c:\MyLib and saves it to c:\MyDoc.
  - *c:\>* ***Javadoc -version -private -d c:\MyDoc -sourcepathlist c:\MySource;c:\YourSource\ -classpath c:\MyLib***

PART-A

1. What is meant by Object Oriented Programming?

OOP is a method of programming in which programs are organized as cooperative collections of objects. Each object is an instance of a class and each class belong to a hierarchy.

2. What are all the advantages of object oriented programming?

- Simplicity
- Re-Usability
- Modularity
- Modifiability
- Extensibility
- Maintainability

3. Differentiate Procedure Oriented Programming and Object Oriented Programming.

Sl.No	Procedure Oriented Programming	Object Oriented Programming
1	Emphasis on Procedure	Emphasis is on data rather than Procedure
2	Programs are divided into functions	Programs are divided into Objects
3	Data are sharable	Data are not sharable
4	Structured Programming	Object Oriented Programming
5	Top-Down Approach	Bottom-Up Approach

4. Define objects and classes in Java.

An object is an instance of class. The object represents the real world entity.

Class is a collection of data and the function that manipulate the data.

5. What is a class?

Class is a template for a set of objects that share a common structure and a common behavior.

6. What are the key characteristics of objects?

Object is an instance of class. Object are runtime entity. Using object of a class the member variable and member function of a class can be accesses.

7. What is meant by data abstraction?(April/May 2015)

It refers to the act of representing essential features without including the background details. Since classes use the concept of abstraction, they are known as Abstract Data types.

8. What is meant by data encapsulation?

Wrapping up of data and functions into a single unit is known as Encapsulation. The data is not accessible to the outside world. Only functions those wrapped inside the class can access it.

9. Define Inheritance.

The properties of base class will be reused in derived class is called inheritance. It permits code reusability. Reusability saves time in program.

10. Define polymorphism

It Means the ability to take more than one form.

The behavior may depend upon the types of data used.

1. Compile time polymorphism. (Function overloading & Operator Overloading)
2. Run time polymorphism. (Virtual Function)

11. Differentiate between object and a class.

Class	Object
Class is the collection of objects of similar types.	Objects are instance of class
Class has collection of member function and data members	Objects are variables of type class

12. What are the features of Java? (N/D 17)

i. Simple. ii. Object Oriented. iii. Platform Independent. iv. Robust. v. Multithreaded. vi. Secure.

13. What is meant by Platform?

Platform is the hardware or system software environment in which your program runs. Most platforms are described as a combination of hardware and operating system.

14. Why is java Language calles as 'robust'? or Java is robust comment. (A/M 16)

Java is called robust because due to JVM can be used on multiple platforms. Secondly it has good mechanism for error checking, automatic garbage collection, strong memory management technique.

15. "Java is platform independent language". Comment. (N/D 16)

Java achieves the platform independence feature due to JVM. The JVM takes the byte code as input and interprets it and execute it for the required Operating System platform.

Due to byte code java program can be executed on any platform with the help of JVM.

16. What is JVM ? (A/M 15)

JVM Stands for Java Virtual Machine. It is Collection of software and program components. Java compiler generates the byte code JVM takes the byte code as input and produce the corresponding machine code. Java achieves the platform independence feature due to JVM.

17. What is byte code? Mention its advantage. (A/M 15) (M/J 16) (A/M 17)

Byte code is an intermediate form of Java programs. By compiling the Java program we can get the byte code which can be executed by JVM. The byte code consist of optimized set of instructions that are not specific to any particular processor.

18. Define Garbage Collection in Java?

Garbage Collection also referred as automatic memory management. Periodically frees the memory used by objects that are no longer needed. The garbage collector in java scans dynamic memory areas for objects and marks those that are referenced. After all possible paths to objects are investigated the unreferenced objects are freed.

19. What is finalize() method?

finalize () method is used just before an object is destroyed and can be called just prior to garbage collection.

20. Give two examples of Java modifiers.

In Java access modifiers are used. These are public, private and protected. Public allows classes, methods and data fields accessible from any class. Private allows classes, methods and data fields accessible from with own class.

21. What are the kinds of java variables?

- Instance variables – They are created when the objects are instantiated and therefore they are associated with objects.

- Class variables – They are global to a class and belong to the entire set of the objects that class creates.
- Local variables – They are declared and used within methods.

22. What do you mean by instance variable?

An Object is an instance of a class. The variable that the object contains are called instance variable.

23. What are wrapper classes in Java?

Wrapper class allow primitive data types to be accessed as objects. Ex: Boolean – java.lang.Boolean; int- java.lang.Integer; float- java.lang.Float;

24. How dynamic initialization of variable is achieved in java?

The variable is Java are allowed to get initialized at run time. Such type of initialization is called dynamic initialization.

For example:

```
double a=5.0;
double b=20.0;
double c=Math.sqrt((a+b));
```

Here the expression is evaluated at run time and its square root value is calculated and then the variable c is initialized dynamically.

25. Define the keyword static in Java. (N/D 15)

Class members such as data member and methods can be declared as static. Static members are shared by all objects. Static methods only access the static data members. With the help of class name we can able to access the static member

26. What is static variable and static method?

Static variable is a class variable which value remains constant for the entire class

Static method is the one which can be called with the class itself and can hold only the static variables

27. Write the output produced by the following code fragments. (N/D 15)

```
System.out.println("Result : " 40+30);
System.out.println("Result :" (40+30));
```

The first System.out statement will produce 4030 as an output.

The second System.out statement will produce 70 as an output.

28. What is constructor?

Constructor is special member function whose name is same as class name. Constructor will be executed automatically whenever object is created and memory is allocated for the object. Constructors are used to initialize the data members.

29. What is a default constructor?

Every time an object is created using new() keyword, at least one constructor is called. It is called a default constructor. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any. Default constructor in Java initializes member data variable to default values (numeric values are initialized as 0, booleans are initialized as *false* and references are initialized as *null*).

30. What is meant by parameter passing constructor? Give an example

A Constructor with parameters is known as Parameterized constructor. Parameterized constructor is used to provide different values to the distinct objects.

During object creation the parameters we pass, determine which constructor should get invoked for object initialization. For example, when we create the object like this MyClass obj = new MyClass(123, "Hi"); then the new keyword invokes the Parameterized constructor with int and string parameters (MyClass(int, String)) after object creation.

31. What is meant by copy constructor?

A *copy constructor* is a *constructor* that creates a new object using an existing object of the same class and initializes each instance variable of newly created object with corresponding instance variables of the existing object passed as argument.

32. What is the difference between a constructor and a method?

Sl.No	Constructor	Method
1	Name of the constructor must be the same as the name of the class	There is no restriction on the name of the method.
2	Constructor does not have any return type	Method has return type. If the method does not return anything then it must have void data type
3	Constructors are chained, in the sense that they are called in some specific order.	Methods are not chained. That means invoking of methods depend upon the logic of the program.

33. What is method overloading and method overriding? (N/D 16)

*Method overloading***:** When a method in a class having the same method name with different arguments is said to be method overloading.

*Method overriding***:** When a method in a class having the same method name with same arguments is said to be method overriding.

34. Give the syntax of while statement in Java. (N/D 17)

```
while(condition)
{
Statements
}
```

35. Why does java makes an executable file?

For the distribution of multiple class file over multiple platform Java makes the class files and package them into one .jar file. The JVM can execute this jar file and an executable file is created in order to run the program on any platform.

36. How do we allocate an array dynamically in Java?

Using the new operator memory for the array can be allocated. The syntax is

```
int a[] = new int[10];
```

37. How is keyword 'super' used in Java?

In inheritance, the keyword super is used for two purposes

To call superclass constructor

To call superclass method.

38. Define access specifier. (R-17 N/D 2018)

Java Access Specifiers (also known as Visibility **Specifiers**) regulate **access** to classes, fields and methods in **Java**.These **Specifiers** determine whether a field or method in a class,

can be used or invoked by another method in another class or sub-class. **Access Specifiers** can be used to restrict **access**.

39. What are different types of access modifiers?

- public: Any thing declared as public can be accessed from anywhere.
- private: Any thing declared as private can't be seen outside of its class.
- protected: Any thing declared as protected can be accessed by classes in the same package and subclasses in the other packages.
- default modifier: Can be accessed only to classes in the same package.

40. Distinguish between arrays and strings

Array	String
It can hold any data type elements.	It holds only character type data.
There is no specific end marker for array	String always ends with Null character
The length of an array is specified in [] at the time of declaration.	The length of string is total number of chara plus one null character.
Java implements array of int, double, float and so on	Java implements string using String StringBuffer class

41. What is the difference between the string and StringBuffer classes?

String class supports constant strings whereas StringBuffer class supports growable and modifiable strings.

42. What is the difference between this() and super()?

this() can be used to invoke a constructor of the same class whereas super() can be used to invoke a super class constructor.

43. What are packages? (A/M 17)

Package is a collection of related classes and interfaces. The main advantage of using package is that the *classes can be arranged systematically*.

44. What are the package of Java?

- Java API Packages
- User Defined Packages

45. List any four JavaDoc comments.

Tags	Description
@author	This can be used in the class level comment. It describe the name of the author who is writing the document
@param	This tag describes the name of th method used in the class.
@return	This tag describe the return tag of the method.
@throws	This tag describe the exertion that can be thrown by the method
@exception	This tag describe the exception

46. Write about API Packages?

Java API provides a large number of classes grouped into different packages according to functionality.

- Language Package(java.lang)
- Utilities Package(java.util)
- I/O Package(java.io)
- Networking Package(java.net)

- AWT Package(java.awt)

47. Can a Java source file be saved using a name other than the class name? Justify. A/M 2019

Yes, you can save your java source code file with any other name, not same as your main class name but when you comiple it than byte code file name will be same as your main class name. ... And you have compile and run a class with different name other than the filename, If you don't have any public methods in this class

48. What are inline functions? Give example. A/M 2019

If a function is inline, the compiler places a copy of the code of that function at each point where the function is called at compile time.

No, Java does not provide inline functions it is typically done by the JVM at execution time.

CS8391 OBJECT ORIENTED PROGRAMMING

Question bank

UNIT –I Part-A

1. Define Objects and classes in java N/D 2010
2. Define Objects and classes. A/M 2014
3. Define Objects and object variable A/M 2011
4. Define the characteristics of object. A/M 2015
5. Define characteristics of Object. A/M 2016
6. What are the key characteristics of object? A/M 2013
7. What do you meant by instance variable A/M 2012
8. Define class N/D 2011
9. Define constructor. N/D 2012
10. What is constructor? A/M 2015 A/M 2016
11. What is constructor? Give an example. A/M 2014
12. What is a default constructor? A/M 2011
13. What is meant by parameter passing constructor? Give an example N/D 2013
14. Mention the various access specifier supported by object oriented programming.N/D 2015
15. List the access specifiers used in Java. N/D 2014
16. List out the type of arrays. N/D 2012
17. How does one import a single package? N/D 2010
18. What is meant by Java Package? N/D 2014
19. List any four Java Doc comments. N/D 2011
20. Define the term virtual machine. N/D 2013
21. Define reflection. N/D 2014
22. Write a simple java program to find given number is prime or not. N/D 2015
23. Write a Java program to display “hello” three times. A/M 2018
24. Write the static and dynamic binding in java. A/M 2014
25. Why do you need finalize method? A/M 2018

2017 Regulation

1. Define Objects and classes in java N/D 2018
2. Define access specifier. N/D 2018
3. Can a Java source file be saved using a name other than the class name? Justify. A/M 2019
4. What are inline functions? Give example. A/M 2019

PART-B

1. Explain what is OOPS and explain the features of OOPS. (8) N/D 2011
2. Describe the concept of OOP. (5) N/D 2013
3. Explain the basic concept of OOP and mention its limitations. (8) A/M 2014
4. How is OOP different from procedure oriented programming languages? (8) A/M 2013
5. Explain the fundamental of OOPs. Discuss the purpose of finalize method with example.(16) N/D 2015
6. List out the basic concepts of Object oriented programming. (4) A/M 2018
7. Java is called as architecturally neutral language – comment. (6) A/M 2018
8. Explain briefly about Java Virtual Machie. (8) A/M 2014

9. What is class? How do you define a class in java? [4] N/D 2010
10. Write down the technique to design good classes. A/M 2013
11. Define class. (2) A/M 2015
12. Define class (2) A/M 2016

13. What is constructor? What is the use of new method? (4) N/D 2010
14. Explain object constructor and other constructor with an example. (6) A/M 2011
15. Discuss about the usage of constructor with an example. (8) N/D 2011
16. What is constructor? Explain with example. (8) A/M 2014
17. What is meant by constructor? Discuss the types of constructor with example.(16) N/D 2014 2015

18. Explain access specifiers (8) N/D 2012
19. Describe briefly about access specifiers in Java. (8) A/M 2014
20. Write shorts notes on access specifiers. (3) A/M 2015
21. Write short notes on access specifiers. (3) A/M 2016

22. Give the syntax for static method and its initialization. [4] N/D 2010
23. Describe static methods and field used in java (10) A/M 2011
24. Explain the term static field and methods and explain its types with examples. (8) A/M 2015
25. Explain the term static fields and methods and explain its type with example. (8) A/M 2016

26. What is meant by overriding method? Give example. (5) N/D 2013
27. What is overriding? How it is different from overloading. (8) A/M 2014

28. Explain the usage of command line parameter. (8) A/M 2011

29. What is a package? How does a compiler locate packages? (4) N/D 2010
30. Define package. Explain the types of package with its importance. (8) N/D 2011
31. Explain package with example. (16) N/D 2012
32. What is package? How to add a class into a package? Give example. (8) A/M 2013
33. What is meant by package? How it is created and implement in Java. (8) N/D 2013
34. Illustrate the concept of importing packages with a sample program. (6) A/M 2018

35. Explain functions of object wrapper and auto boxing with suitable example. (8) A/M 2011

36. Explain arrays in java. (8) N/D 2010
37. Define array. What is array sorting and explain with an example. (8) A/M 2015
38. Define array. What is array sorting and explain with an example? (8) A/M 2016
39. Write Java program to sort ten names in descending order. (16) A/M 2012
40. Write a Java program to reverse the given number. (6) N/D 2013
41. Write a Java program to find the smallest number in the given list. (8) N/D 2013
42. Write notes on destructor. Develop a simple JAVA program to sort the given numbers in increasing order. (16) N/D 2014
43. Consider the loan processing system in a bank. Identify the classes and objects in the system and list them. (6) A/M 2018
44. Write a java program to find the sum of 'n' numbers stored in an array. (10) A/M 2018

45. Discuss briefly about JavaDoc Comments. (8) A/M 2014
46. State and explain documentation comments in Java. (8) A/M 2015
47. State and explain documentation comments in Java. (8) A/M 2016

2017 Regulation

N/D 2018

1. Explain the characteristics of OOPs. (6)
2. Explain the features and the characteristics of JAVA. (7)
3. What is method? How method is defined? Give example. (6)
4. State the purpose of finalize() method in java? With an example explain how finalize () method can be used in java program. (7)

A/M 2019

5. Discuss the three OOP principles in detail. (7)
6. What are literals? Explain the types of literals supported by java. (6)
7. Explain the selection statements in Java using suitable examples. (7)
8. Write a Java code using do-while loop that counts down to 1 from 10 printing exactly ten lines of "hello". (6)