JAVA PROGRAMMING

UNIT-I

Data Types and Variables: The Simple Types - Literals - Variables - Type Conversion and Casting - Automatic Type Promotion in Expressions -Arrays Strings - Classes and Methods: Class Fundamentals - Declaring Class Objects Constructors - Garbage Collection - The finalize() Method - Overloading Methods - Argument Passing - Recursion - Understanding Static - Access Control--: The main ( ) method.

UNIT- II

Operators: Arithmetic Operators - Bit wise Operators - Relational OperatorsBoolean Logical Operators - The Assignment Operator - The? Operator- The Dot Operator - Operator Precedence - Inheritance, Packages, and Interfaces: Inheritance - Using Super - When Constructors are called - Method Overriding - Abstract Classes - The final Keyword -Packages - Importing Packages - Access ControlInterfaces - Keyword Summary.

UNIT- III

The Language Classes and Interfaces - The Utility Classes and Interfaces - The Input/Output Classes and Interfaces.

UNIT-IV

The Networking Classes and Interfaces - The Java Applet Class and Interfaces.

IT- V

The Abstract Window Toolkit Classes and Interfaces - The Event Classes and Interfaces. .

Text Book :

1."Java - Programmer's Reference", Herbert Schildt

with Joe O'Neil, Tata McGraw Hill, 1998.

Reference Books:

1. "Internet Programming", Kris James Ph.D., and Ke

n Cope, Galgotia Publication, Reprint 2000

2. "Complete Reference", 'Patrick Naughton and Herb

ert Schildt, 3rd Edition,Tata McGraw Hill

Publishing Company Ltd., 199

JAVA PROGRAMMING

UNIT-I

TWO MARK

1.What is java?

Java is a **programming language** and a **platform**. Java is a high level, robust, secured and object-oriented programming language.

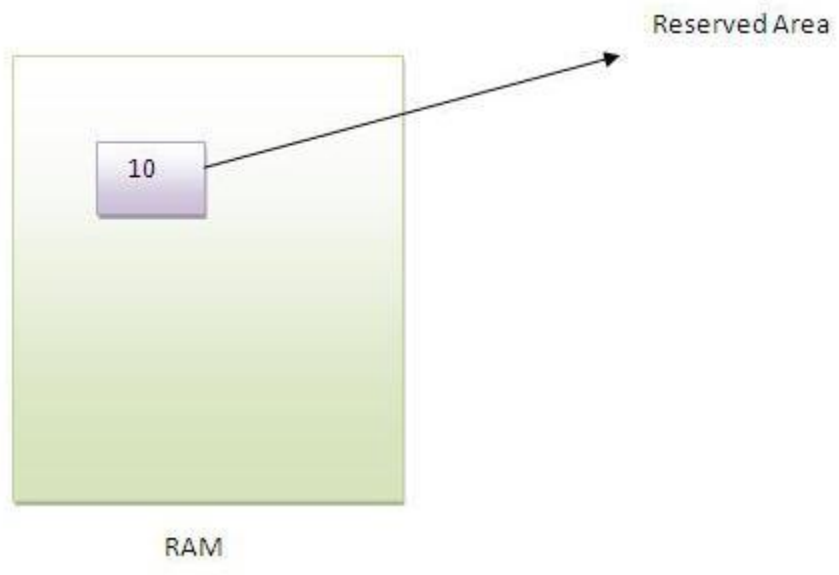
Platform: Any hardware or software environment in which a program runs, is known as a platform. Since Java has its own runtime environment (JRE) and API, it is called platform.

2. List any five features of Java?

Some features include Object Oriented, Platform Independent, Robust, Interpreted, Multi-threaded

3.What is meant by variables?

Variable is name of reserved area allocated in memory.

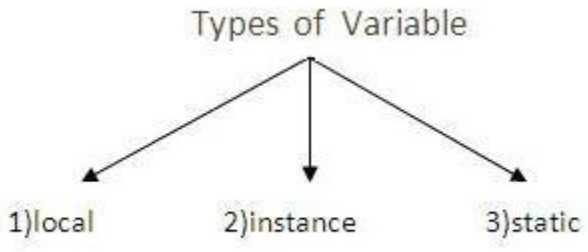


1. int data=50;//Here data is variable

4.Types of Variable

There are three types of variables in java

- local variable
- instance variable
- static variable



5. What is meant by local Variable?

A variable that is declared inside the method is called local variable

6.What is an Object?

Object is an instance of a class. It has state,behaviour and identity. It is also called as an
instance of a class.

7.What is meant by abstraction?

Abstraction defines the essential characteristics of an object that distinguish it from all other kinds of objects. Abstraction provides crisply-defined conceptual boundaries relative to the perspective of the viewer. Its the process of focussing on the essential characteristics of an object. Abstraction is one of the fundamental elements of the object model.

8..What is meant by Encapsulation?

Encapsulation is the process of compartmentalising the elements of an abtraction that defines the structure and behaviour. Encapsulation helps to separate the contractual interface of an abstraction and implementation

9. What is meant by instance Variable?

A variable that is declared inside the class but outside the method is called instance variable . It is not declared as static.

10.What is Static variable?

A variable that is declared as static is called static variable. It cannot be local.

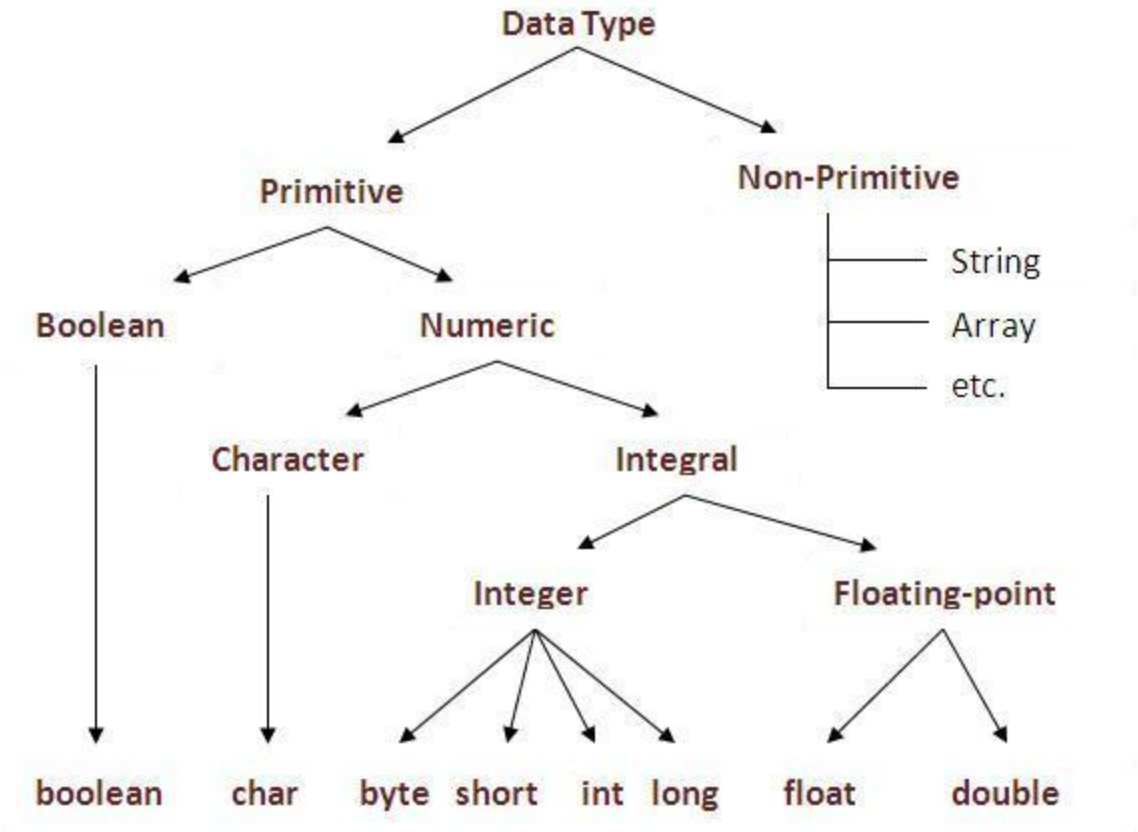
Example to understand the types of variables

```
1. class A{
2. int data=50;//instance variable
3. static int m=100;//static variable
4. void method(){
5. int n=90;//local variable
6. }
7. }//end of class
```

11.What are the Data Types in Java?

In java, there are two types of data types

- primitive data types
- non-primitive data types



12.What is Type Conversion and Casting?

If the two types are compatible, then Java will perform the conversion automatically.

For example, assign an int value to a long variable.

For incompatible types we must use a cast.

Casting is an explicit conversion between incompatible types.

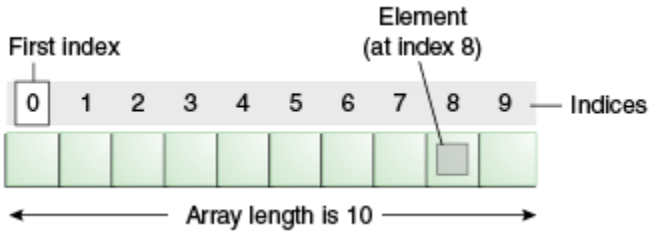
13. How many conditions in Java's Automatic Conversions?

An automatic type conversion will be used if the following two conditions are met:

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

14.Define Arrays

An *array* is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed.



An array of 10 elements.

15.Define class.

A *class*--the basic building block of an object-oriented language such as Java--is a template that describes the data and behavior associated with *instances* of that class. When you *instantiate* a class you create an *object* that looks and feels like other instances of the same class.

```
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); //Display the string.
    }
}
```

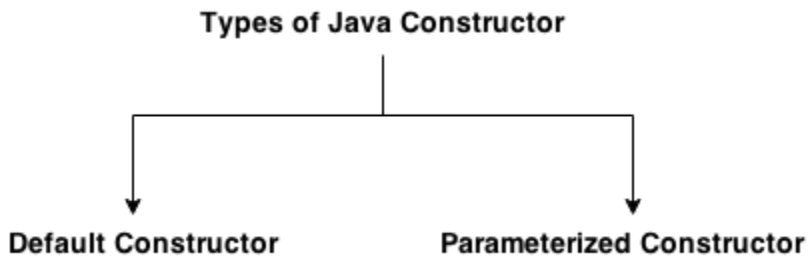
16. Define Constructor?

Constructor in java is a *special type of method* that is used to initialize the object. Java constructor is *invoked at the time of object creation*. It constructs the values i.e. provides data for the object that is why it is known as constructor.

17. What are the Types of java constructors?

There are two types of constructors:

1. Default constructor (no-arg constructor)
2. Parameterized constructor



18. What is meant by garbage collection?

It frees memory allocated to objects that are not being used by the program any more - hence the name "garbage". For example:

```
public static Object otherMethod(Object obj) {
   return new Object();
}

public static void main(String[] args) {
   Object myObj = new Object();
   myObj = otherMethod(myObj);
   // ... more code ...
}
```

19. Define Method Overloading .

If a class have multiple methods by same name but different parameters, it is known as **Method Overloading**.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

20.What is recursion?

Java supports recursion. Recursion is the process of defining something in terms of itself. As it relates to java programming, recursion is the attribute that allows a method to call itself. A method that calls itself is said to be recursive.

5 MARK

1.Write about the features of Java.

In an object-oriented system, a class is a collection of data and methods that operate on that data. Taken together, the data and methods describe the state and behavior of an object. Classes are arranged in a hierarchy, so that a subclass can inherit behavior from its superclass.

Distributed

- It has a spring-like transparent RPC system
- Now uses mostly tcp-ip based protocols like ftp & http

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.

- rapid turn-around development
- Software author is protected, since binary byte streams are downloaded and not the source code

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; this is repeated at runtime for flexibilty and to check consistency

Secure

 access restrictions are enforced (public, private)
 byte codes are verified, which copes with the threat of a hostile compiler

Architecture-Neutral

 compiler generates bytecodes, which have nothing to do with a particular computer architecture
 easy to interpret on any machine

Portable

Java goes further than just being architecture-neutral:
 no "implementation dependent" notes in the spec (arithmetic and evaluation order)
 standard libraries hide system differences
 the Java environment itself is also portable: the portability boundary is POSIX compliant .

High-Performance

Java is an interpreted language, so it will never be as fast as a compiled language as C or C++. In fact, it is about 20 times as slow as C. However, this speed is more than enough to run interactive, GUI and network-based applications, where the application is often idle, waiting for the user to do something, or waiting for data from the network.

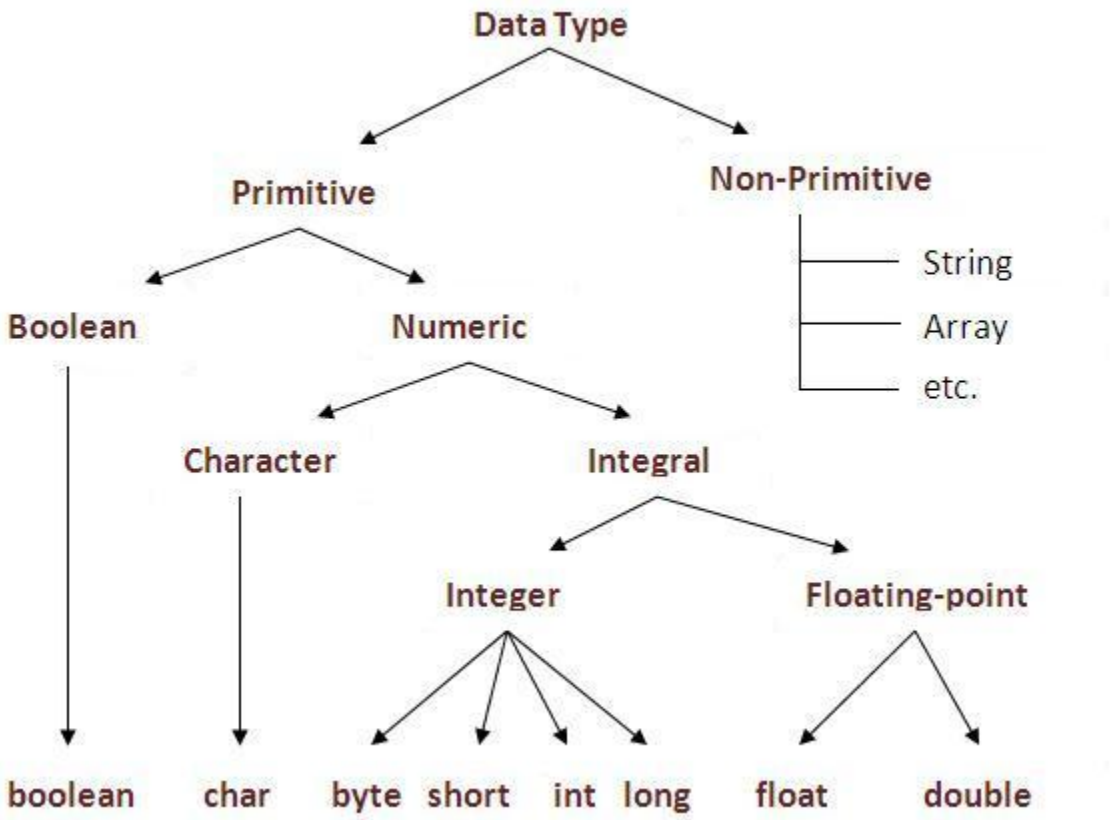
Dynamic

Java was designed to adapt to an evolving environment:
 Even after binaries have been released, they can adapt to a changing environment
 Java loads in classes as they are needed, even from across the network

2.What are the different Data Types in Java

In java, there are two types of data types

- primitive data types
- non-primitive data types



Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte

8 byte

double 0.0d

3.Explain about local variables.

Local variables:

- Local variables are declared in methods, constructors, or blocks.
- Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor or block.
- Access modifiers cannot be used for local variables.
- Local variables are visible only within the declared method, constructor or block.
- Local variables are implemented at stack level internally.
- There is no default value for local variables so local variables should be declared and an initial value should be assigned before the first use.

Example:

Here, *age* is a local variable. This is defined inside *pupAge()* method and its scope is limited to this method only.

```
public class Test{
  public void pupAge(){
    int age = 0;
    age = age + 7;
    System.out.println("Puppy age is : " + age);
  }

  public static void main(String args[]){
    Test test = new Test();
    test.pupAge();
  }
```

4.Explain about Classes in Java

A class is a blue print from which individual objects are created.

A sample of a class is given below:

```
public class Dog{
  String breed;
  int age;
  String color;

  void barking(){
  }

  void hungry(){
  }

  void sleeping(){
  }
}
```

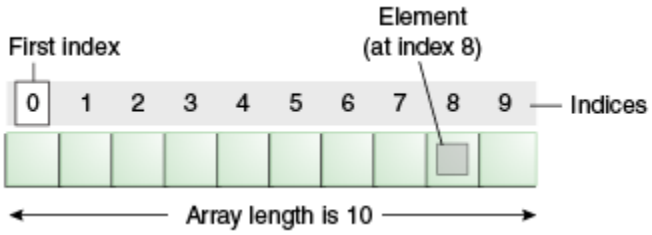
A class can contain any of the following variable types.

- **Local variables:** Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
- **Instance variables:** Instance variables are variables within a class but outside any method. These variables are instantiated when the class is loaded. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.
- **Class variables:** Class variables are variables declared with in a class, outside any method, with the static keyword.

5.Write short notes on Arrays

Arrays

An *array* is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed. You have seen an example of arrays already, in the main method of the "Hello World!" application. This section discusses arrays in greater detail.



An array of 10 elements.

Each item in an array is called an *element*, and each element is accessed by its numerical *index*..

You can place strings of text into arrays. This is done in the same way as for integers:

String[ ] aryString = new String[5] ;

aryString[0]="This";
aryString[1]="is";
aryString[2]="a";

aryString[3]="string";
aryString[4] = "array";

The code above sets up a string array with 5 positions. Text is then assigned to each position in the array.

example

```
class ArrayDemo {
    public static void main(String[] args) {
        // declares an array of integers
        int[] anArray;

        // allocates memory for 10 integers
        anArray = new int[10];

        // initialize first element
        anArray[0] = 100;
        // initialize second element
        anArray[1] = 200;
        // and so forth
        anArray[2] = 300;
        anArray[3] = 400;
        anArray[4] = 500;
        anArray[5] = 600;
        anArray[6] = 700;
        anArray[7] = 800;
        anArray[8] = 900;
        anArray[9] = 1000;

        System.out.println("Element at index 0: "
                           + anArray[0]);
        System.out.println("Element at index 1: "
                           + anArray[1]);
        System.out.println("Element at index 2: "
                           + anArray[2]);
        System.out.println("Element at index 3: "
                           + anArray[3]);
        System.out.println("Element at index 4: "
                           + anArray[4]);
        System.out.println("Element at index 5: "
                           + anArray[5]);
```

```
        System.out.println("Element at index 6: "
                        + anArray[6]);
        System.out.println("Element at index 7: "
                        + anArray[7]);
        System.out.println("Element at index 8: "
                        + anArray[8]);
        System.out.println("Element at index 9: "
                        + anArray[9]);
    }
}
```

6. Program to find the factorial of a number using recursion

```
import java.io.*;
class recurs
{
public static void main(String arg[])
{
int num=0;
DataInputStream ins=new DataInputStream(System.in);
try
{
System.out.print(
"Enter the no.");
num=Integer.parseInt(ins.readLine());
}
catch(IOException e)
{}
System.out.print("Factorial is "+fact(num));
}
static int fact(int n)
{
if(n==1)
return 1;
```

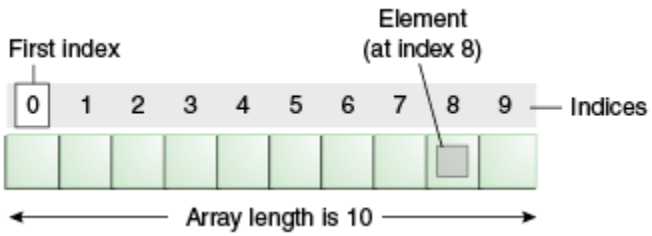
```
else
return (n*fact(n-1));
}
}
```

10 MARK

1.Write short notes on Arrays

Arrays

An *array* is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed. You have seen an example of arrays already, in the main method of the "Hello World!" application. This section discusses arrays in greater detail.



An array of 10 elements.

Each item in an array is called an *element*, and each element is accessed by its numerical *index*..You can place strings of text into arrays. This is done in the same way as for integers:

```
String[ ] aryString = new String[5] ;

aryString[0]="This";
aryString[1]="is";
aryString[2]="a";
aryString[3]="string";
aryString[4] = "array";
```

The code above sets up a string array with 5 positions. Text is then assigned to each position in the array.

example

```
class ArrayDemo {
    public static void main(String[] args) {
        // declares an array of integers
        int[] anArray;

        // allocates memory for 10 integers
        anArray = new int[10];

        // initialize first element
        anArray[0] = 100;
        // initialize second element
        anArray[1] = 200;
        // and so forth
        anArray[2] = 300;
        anArray[3] = 400;
        anArray[4] = 500;
        anArray[5] = 600;
        anArray[6] = 700;
        anArray[7] = 800;
        anArray[8] = 900;
        anArray[9] = 1000;

        System.out.println("Element at index 0: "
                           + anArray[0]);
        System.out.println("Element at index 1: "
                           + anArray[1]);
        System.out.println("Element at index 2: "
                           + anArray[2]);
        System.out.println("Element at index 3: "
                           + anArray[3]);
```

```
        System.out.println("Element at index 4: "
                           + anArray[4]);
        System.out.println("Element at index 5: "
                           + anArray[5]);
        System.out.println("Element at index 6: "
                           + anArray[6]);
        System.out.println("Element at index 7: "
                           + anArray[7]);
        System.out.println("Element at index 8: "
                           + anArray[8]);
        System.out.println("Element at index 9: "
                           + anArray[9]);
    }
}
```

The output from this program is:

```
Element at index 0: 100
Element at index 1: 200
Element at index 2: 300
Element at index 3: 400
Element at index 4: 500
Element at index 5: 600
Element at index 6: 700
Element at index 7: 800
Element at index 8: 900
Element at index 9: 1000
```

Copying Arrays

The System class has an arraycopy method that you can use to efficiently copy data from one array into another:

```
public static void arraycopy(Object src, int srcPos,
                             Object dest, int destPos, int length)
```

The two Object arguments specify the array to copy *from* and the array to copy *to*. The three int arguments specify the starting position in the source array, the starting position in the destination array, and the number of array elements to copy.

The following program, ArrayCopyDemo, declares an array of char elements, spelling the word "decaffeinated." It uses the System.arraycopy method to copy a subsequence of array components into a second array:

```
class ArrayCopyDemo {
    public static void main(String[] args) {
        char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e',
                                        'i', 'n', 'a', 't', 'e', 'd' };
        char[] copyTo = new char[7];

        System.arraycopy(copyFrom, 2, copyTo, 0, 7);
        System.out.println(new String(copyTo));
    }
}
```

The output from this program is:

Caffeine

2.Explain about the Constructors.

A **constructor** is a bit of code that allows you to create objects from a class. You call the constructor by using the keyword new, followed by the name of the class, followed by any necessary parameters. For example, if you have a Dog class, you can create new objects of this type by saying new Dog().

The syntax for a constructor is:

```
access NameOfClass(parameters) {
        initialization code
```

```
}
```

where

- access is one of public, protected, "package" (default), or private;
- *NameOfClass* must be identical to the name of the class in which the constructor is defined; and
- the *initialization code* is ordinary Java declarations and statements.

The term "constructor" is misleading since, as soon as you enter the constructor, the new object has actually been created for you. The job of the constructor is to ensure that the new object is in a valid state, usually by giving initial values to the instance variables of the object. So a "constructor" should really be called an "initializer."

Every class has at least one constructor. There are two cases:

1. If you do not write a constructor for a class, Java generates one for you. This generated constructor is called a **default constructor**. It's not visible in your code, but it's there just the same. If you could see it, it would look like this (for the class Dog):

   ```
   public Dog() { }
   ```

   Notice that this default constructor takes no arguments and has a body that does nothing.
2. If you *do* write a constructor for your class, Java does *not* generate a default constructor. This could be a problem if you have pre-existing code that uses the default constructor.

Example constructors:

```
class Dog extends Animal {
   String name;
   String breed;
```

```
    // first constructor
    public Dog(String s) {
        name = s;
        breed = "unknown";
    }

    // second constructor
    public Dog(String name, String breed) {
        this.name = name;
        this.breed = breed;
    }
}
```

To avoid having to use different names for the same thing, the second constructor uses a simple trick. The parameter names are the same as the names of some instance variables; to distinguish the two, this.*variable* refers to the instance variable, while *variable* refers to the parameter. This naming convention is very commonly used.

3.Method Overloading in Java with examples

OOPs Concept

Method Overloading is a feature that allows a class to have two or more methods having same name, if their argument lists are different. In the last tutorial we discussed constructor overloading that allows a class to have more than one constructors having different argument lists.

Argument lists could differ in –

1. Number of parameters.
2. Data type of parameters.
3. Sequence of Data type of parameters.

Method overloading is also known as **Static Polymorphism**.

Points to Note:

1. Static Polymorphism is also known as compile time binding or early binding.
2. Static binding happens at compile time. Method overloading is an example of static binding where binding of method call to its definition happens at Compile time.

Method Overloading examples:

As discussed above, method overloading can be done by having different argument list. Lets see examples of each and every case.

Example 1: Overloading – Different Number of parameters in argument list

When methods name are same but number of arguments are different.

```
class DisplayOverloading
{
   public void disp(char c)
   {
       System.out.println(c);
   }
   public void disp(char c, int num)
   {
       System.out.println(c + " "+num);
   }
}
class Sample
{
  public static void main(String args[])
  {
     DisplayOverloading obj = new DisplayOverloading();
     obj.disp('a');
     obj.disp('a',10);
  }
}
```

Output:

```
a
a 10
```

In the above example – method disp() has been overloaded based on the number of arguments – We have two definition of method disp(), one with one argument and another with two arguments.

Example 2: Overloading – Difference in data type of arguments

In this example, method disp() is overloaded based on the data type of arguments – Like example 1 here also, we have two definition of method disp(), one with char argument and another with int argument.

```
class DisplayOverloading2
{
   public void disp(char c)
   {
       System.out.println(c);
   }
   public void disp(int c)
   {
     System.out.println(c );
   }
}

class Sample2
{
   public static void main(String args[])
   {
      DisplayOverloading2 obj = new DisplayOverloading2();
      obj.disp('a');
      obj.disp(5);
   } }
```

Output:

a

5

Example3: Overloading – Sequence of data type of arguments

Here method disp() is overloaded based on sequence of data type of arguments – Both the methods have different sequence of data type in argument list. First method is having argument list as (char, int) and second is having (int, char). Since the sequence is different, the method can be overloaded without any issues.

```
class DisplayOverloading3
{
  public void disp(char c, int num)
  {
     System.out.println("I’m the first definition of method disp");
  }
  public void disp(int num, char c)
  {
     System.out.println("I’m the second definition of method disp" );
  }
}
class Sample3
{
  public static void main(String args[])
  {
     DisplayOverloading3 obj = new DisplayOverloading3();
     obj.disp('x', 51 );
     obj.disp(52, 'y');
  }
}
```

Output:

I’m the first definition of method disp

I’m the second definition of method disp

UNIT-II

TWO MARKS

1.Explain the usage of Java packages

This is a way to organize files when a project consists of multiple modules. It also helps resolve naming conflicts when different packages have classes with the same names. Packages access level also allows you to protect data from being used by the nonauthorized classes.

2.What is method overloading and method overriding?

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.

3.What gives java it’s “write once and run anywhere” nature?

All Java programs are compiled into class files that contain bytecodes. These byte codes can be run in any platform and hence java is said to be platform independent.

4.Define arithmetic operators in java?

Arithmetic operators perform the same basic operations you would expect if you used them in mathematics (with the exception of the percentage sign). They take two operands and return the result of the mathematical calculation.

Java has five arithmetic operators:

+ to add two numbers together or concatenate two Strings.

- to subtract one number from another.

* to multiply one number by another.

/ to divide one number by another.

% to find the remainder from dividing one number by another.

5.Define bitwise operator.

Bitwise operators perform operations on the bits of their operands. The operands can only be byte, short, int, long or char data types. For example, if an operand is the number 48, the bitwise operator will perform its operation on the binary representation of 48 (i.e., 110000).

There are three binary logical operators:

& performs a logical AND operation.

| performs a logical OR operation.

^ performs a logical XOR operation.

6.Precedence order.

When two operators share an operand the operator with the higher *precedence* goes first. For example, 1 + 2 * 3 is treated as 1 + (2 * 3), whereas 1 * 2 + 3 is treated as (1 * 2) + 3 since multiplication has a higher precedence than addition.

7.Definition - What does Inheritance mean?

Inheritance is a mechanism wherein a new class is derived from an existing class. In Java, classes may inherit or acquire the properties and methods of other classes.A class derived from another class is called a subclass, whereas the class from which a subclass is derived is called a superclass. A subclass can have only one superclass, whereas a superclass may have one or more subclasses.

8.What is finalize() method?

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

9.What is the difference between String and String Buffer?

a) String objects are constants and immutable whereas StringBuffer objects are not.

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

10. What is an Abstract Class?

Abstract class is a class that has no instances. An abstract class is written with the expectation that its concrete subclasses will add to its structure and behaviour, typically by implementing its abstract operations.

11.What is an Interface?

Interface is an outside view of a class or object which emphaizes its abstraction while hiding its structure and secrets of its behaviour.

5 MARK

12.The Arithmetic Operators:

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators:Assume integer variable A holds 10 and variable B holds 20, then:

Show Examples

Operator	Description	Example
+	Addition - Adds values on either side of the operator	A + B will give 30

-	Subtraction - Subtracts right hand operand from left hand operand	A - B will give -10
*	Multiplication - Multiplies values on either side of the operator	A * B will give 200
/	Division - Divides left hand operand by right hand operand	B / A will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	B % A will give 0
++	Increment - Increases the value of operand by 1	B++ gives 21
--	Decrement - Decreases the value of operand by 1	B-- gives 19

The Relational Operators:

There are following relational operators supported by Java language

Assume variable A holds 10 and variable B holds 20, then:

Show Examples

Operator	**Description**	**Example**
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.

<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes (A <= B) is true. true.

13.Operator precedence

Java has well-defined rules for specifying the order in which the operators in an expression are evaluated when the expression has several operators. For example, multiplication and division have a higher precedence than addition and subtraction. Precedence rules can be overridden by explicit parentheses.

Precedence order.

When two operators share an operand the operator with the higher *precedence* goes first. For example, 1 + 2 * 3 is treated as 1 + (2 * 3), whereas 1 * 2 + 3 is treated as (1 * 2) + 3 since multiplication has a higher precedence than addition.

Associativity.

When an expression has two operators with the same precedence, the expression is evaluated according to its *associativity*. For example x = y = z = 17 is treated as x = (y = (z = 17)), leaving all three variables with the value 17, since the = operator has right-to-left associativity (and an assignment statement evaluates to the value on the right hand side). On the other hand, 72 / 2 / 3 is treated as (72 / 2) / 3 since the / operator has left-to-right associativity.

Precedence and associativity of Java operators.

The table below shows all Java operators from highest to lowest precedence, along with their associativity. Most programmers do not memorize them all, and even those that do still use parentheses for clarity.

Operator	Description	Level	Associativity
[] . ()	access array element access object member invoke a method	1	left to right

++ --	post-increment post-decrement		
++ -- + - ! ~	pre-increment pre-decrement unary plus unary minus logical NOT bitwise NOT	2	right to left
() new	cast object creation	3	right to left
* / %	multiplicative	4	left to right
+ - +	additive string concatenation	5	left to right
<< >> >>>	shift	6	left to right
< <= > >= instanceof	relational type comparison	7	left to right
== !=	equality	8	left to right
&	bitwise AND	9	left to right
^	bitwise XOR	10	left to right
|	bitwise OR	11	left to right
&&	conditional AND	12	left to right
||	conditional OR	13	left to right
?:	conditional	14	right to left

= += -= *= /= %= &= ^= |= <<= >>= >>>=	assignment	15	right to left

There is no explicit operator precedence table in the Java Language Specification and different tables on the web and in textbooks disagree in some minor ways.

If a class inherits a method from its super class, then there is a chance to override the method provided that it is not marked final.

The benefit of overriding is: ability to define a behavior that's specific to the subclass type which means a subclass can implement a parent class method based on its requirement.

In object-oriented terms, overriding means to override the functionality of an existing method.

Example:

Let us look at an example.

```
class Animal{

  public void move(){
    System.out.println("Animals can move");
  }
}

class Dog extends Animal{
```

```
  public void move(){
    System.out.println("Dogs can walk and run");
  }
}

public class TestDog{

  public static void main(String args[]){
    Animal a = new Animal(); // Animal reference and object
    Animal b = new Dog(); // Animal reference but Dog object

    a.move();// runs the method in Animal class

    b.move();//Runs the method in Dog class
  }
}
```

This would produce the following result:

```
Animals can move
Dogs can walk and run
```

In the above example, you can see that the even though **b** is a type of Animal it runs the move method in the Dog class. The reason for this is: In compile time, the check is made on the reference type. However, in the runtime, JVM figures out the object type and would run the method that belongs to that particular object.

Therefore, in the above example, the program will compile properly since Animal class has the method move. Then, at the runtime, it runs the method specific for that object.

Consider the following example :

```
class Animal{

  public void move(){
    System.out.println("Animals can move");
  }
}

class Dog extends Animal{

  public void move(){
    System.out.println("Dogs can walk and run");
  }
  public void bark(){
    System.out.println("Dogs can bark");
  }
}

public class TestDog{

  public static void main(String args[]){
    Animal a = new Animal(); // Animal reference and object
    Animal b = new Dog(); // Animal reference but Dog object

    a.move();// runs the method in Animal class
    b.move();//Runs the method in Dog class
    b.bark();
  } }
```

This would produce the following result:

```
TestDog.java:30: cannot find symbol
symbol  : method bark()
location: class Animal
            b.bark();
```

^

This program will throw a compile time error since b's reference type Animal doesn't have a method by the name of bark.

14.Rules for method overriding*:*

- The argument list should be exactly the same as that of the overridden method.
- The return type should be the same or a subtype of the return type declared in the original overridden method in the superclass.
- The access level cannot be more restrictive than the overridden method's access level. For example: if the superclass method is declared public then the overridding 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.
- A subclass within the same package as the instance's superclass can override any superclass method that is not declared private or final.
- A subclass in a different package can only override the non-final methods declared public or protected.
- An overriding method can throw any uncheck exceptions, regardless of whether the overridden method throws exceptions or not. However the overriding method should not throw checked exceptions that are new or broader than the ones declared by the overridden method. The overriding method can throw narrower or fewer exceptions than the overridden method.
- Constructors cannot be overridden.

15.Write a program for inheritance

Add and Subtract Numbers using Class and Inheritance

```
class pqr
{
int a, b;
void getdata()
```

```
{
a=10;
b=20;
}
}
class sum extends pqr
{
int sum;
void sum()
{
sum = a + b;
System.out.print("
\
nSum = " + sum);
}
}
class subt extends sum
{
i
nt subt;
void subtract()
{
subt = a
-
b;
W3
Professors.Com
System.out.print("
\
nSubtraction = " + subt + "
\
n");
}
}
```

```
class abc
{
public static void main(String args[])
{
subt obj = new subt();
obj.getdata();
obj.sum();
obj.subtract();
}
}
```

16. Program to implement constructor overloading

```
class Sum
{
int x, y, z;
String p, q;
Sum(int a, int b)
{
x=a;
y=b;
z=a+b;
System.out.print("The sum of numbers is: " + z);
}
Sum (String h, String i)
{
p=h;
q=i;
System.out.p
rintln("Hi, " + p + " " + q);
}
}
class constr
{
```

```
public static void main(String args[])
{
Sum k = new Sum("Rakesh", "Kumar");
Sum d = new Sum(10, 20);
}
}
```

10 MARK

17.Explain about Arithmetic operators

java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups:

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- Misc Operators

The Arithmetic Operators:

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators:

Assume integer variable A holds 10 and variable B holds 20, then:

Show Examples

Operator	Description	Example
+	Addition - Adds values on either side of the operator	A + B will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	A - B will give -10

*	Multiplication - Multiplies values on either side of the operator	A * B will give 200
/	Division - Divides left hand operand by right hand operand	B / A will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	B % A will give 0
++	Increment - Increases the value of operand by 1	B++ gives 21
--	Decrement - Decreases the value of operand by 1	B-- gives 19

The Relational Operators:

There are following relational operators supported by Java language

Assume variable A holds 10 and variable B holds 20, then:

Show Examples

Operator	**Description**	**Example**
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

The Bitwise Operators:

Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte.

Bitwise operator works on bits and performs bit-by-bit operation. Assume if a = 60; and b = 13; now in binary format they will be as follows:

a = 0011 1100

b = 0000 1101

---

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a = 1100 0011

The following table lists the bitwise operators:

Assume integer variable A holds 60 and variable B holds 13 then:

Show Examples

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011

		0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>>	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

The Logical Operators:

The following table lists the logical operators:

Assume Boolean variables A holds true and variable B holds false, then:

Show Examples

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

The Assignment Operators:

There are following assignment operators supported by Java language:

Show Examples

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2

|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

Misc Operators

There are few other operators supported by Java Language.

Conditional Operator (? :):

Conditional operator is also known as the ternary operator. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide which value should be assigned to the variable. The operator is written as:

variable x = (expression) ? value if true : value if false

Following is the example:

```
public class Test {

  public static void main(String args[]){
    int a , b;
    a = 10;
    b = (a == 1) ? 20: 30;
    System.out.println( "Value of b is : " +  b );

    b = (a == 10) ? 20: 30;
    System.out.println( "Value of b is : " + b );
  }
}
```

This would produce the following result:

```
Value of b is : 30
Value of b is : 20
```

instanceof Operator:

This operator is used only for object reference variables. The operator checks whether the object is of a particular type(class type or interface type). instanceof operator is wriiten as:

(Object reference variable) instanceof (class/interface type)

If the object referred by the variable on the left side of the operator passes the IS-A check for the class/interface type on the right side, then the result will be true. Following is the example:

```
public class Test {

  public static void main(String args[]){
    String name = "James";
    // following will return true since name is type of String
    boolean result = name instanceof String;
    System.out.println( result );
  }
}
```

This would produce the following result:

```
true
```

This operator will still return true if the object being compared is the assignment compatible with the type on the right. Following is one more example:

```
class Vehicle {}

public class Car extends Vehicle {
  public static void main(String args[]){
    Vehicle a = new Car();
    boolean result =  a instanceof Car;
    System.out.println( result );
```

```
  }
}
```

This would produce the following result:true

18.Precedence of Java Operators:

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator:

For example, x = 7 + 3 * 2; here x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	**Operator**	**Associativity**
Postfix	() [] . (dot operator)	Left to right
Unary	++ - - ! ~	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	>> >>> <<	Left to right
Relational	> >= < <=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left

inheritance can be defined as the process where one object acquires the properties of another. With the use of inheritance the information is made manageable in a hierarchical order.

When we talk about inheritance, the most commonly used keyword would be **extends** and **implements**. These words would determine whether one object IS-A type of another. By using these keywords we can make one object acquire the properties of another object.

IS-A Relationship:

IS-A is a way of saying : This object is a type of that object. Let us see how the **extends** keyword is used to achieve inheritance.

```
public class Animal{
}

public class Mammal extends Animal{
}

public class Reptile extends Animal{
}

public class Dog extends Mammal{
}
```

Now, based on the above example, In Object Oriented terms, the following are true:

- Animal is the superclass of Mammal class.
- Animal is the superclass of Reptile class.
- Mammal and Reptile are subclasses of Animal class.
- Dog is the subclass of both Mammal and Animal classes.

Now, if we consider the IS-A relationship, we can say:

- Mammal IS-A Animal

- Reptile IS-A Animal
- Dog IS-A Mammal
- Hence : Dog IS-A Animal as well

With use of the extends keyword the subclasses will be able to inherit all the properties of the superclass except for the private properties of the superclass.

We can assure that Mammal is actually an Animal with the use of the instance operator.

Example:

```
public class Dog extends Mammal{

  public static void main(String args[]){

    Animal a = new Animal();
    Mammal m = new Mammal();
    Dog d = new Dog();

    System.out.println(m instanceof Animal);
    System.out.println(d instanceof Mammal);
    System.out.println(d instanceof Animal);
  }
}
```

This would produce the following result:

```
true
true
true
```

Since we have a good understanding of the **extends** keyword let us look into how the **implements** keyword is used to get the IS-A relationship.

The **implements** keyword is used by classes by inherit from interfaces. Interfaces can never be extended by the classes.

Example:

```
public interface Animal {}

public class Mammal implements Animal{
}

public class Dog extends Mammal{
```

Unit III
2 MARK

1.What is Class in Java programming?
A class can be defined as a template/blue print that describes the behaviors/states that object of its type support.

2.What is nested class?

If all the methods of a inner class is static then it is a nested class.

3.What is inner class?

If the methods of the inner class can only be accessed via the instance of the inner class, then it is called inner class.

4. What is Class.forName() does and how it is useful?

It loads the class into the ClassLoader. It returns the Class. Using that you can get the

instance (―class-instance‖.newInstance()).

5. What are the methods in Object?

clone, equals, wait, finalize, getClass, hashCode, notify, notifyAll, toString

6. What is an Interface?
Interface is an outside view of a class or object which emphaizes its
abstraction while
hiding its structure and secrets of its behaviour.

7. What is the useful of Interfaces?
a) Declaring methods that one or more classes are expected to implement
b) Capturing similarities between unrelated classes without forcing a class relationship.
c) Determining an object's programming interface without revealing the
actual body of the class.

8. What is a cloneable interface and how many methods does it contain?
It is not having any method because it is a TAGGED or MARKER interface.

9.Mention the JDK I/O packages in Java.

JDK has two sets of I/O packages:

1. the Standard I/O (in package java.io), introduced since JDK 1.0 for stream-based I/O, and

2. the New I/O (in packages java.nio), introduced in JDK 1.4, for more efficient buffer-based I/O.

10. What is stream?

A *stream* is a sequential and contiguous one-way flow of data.Java does not differentiate between the various types of data sources or sinks (e.g., file or network) in stream I/O.

11.Mention the stream I/O operations.

Stream I/O operations involve three steps:

1. *Open* an input/output stream associated with a physical device (e.g., file, network, console/keyboard), by constructing an appropriate I/O stream instance.
2. *Read* from the opened input stream until "end-of-stream" encountered, or *write* to the opened output stream (and optionally flush the buffered output).
3. *Close* the input/output stream.

12. What is Byte Stream?

Java byte streams are used to perform input and output of 8-bit bytes as FileInputStream and FileOutputStream.

13. Define character stream.

The Java platform stores character values using Unicode conventions. Character stream I/O automatically translates this internal format to and from the local character set.

14. What is scanner?

The Scanner class has a method called nextLine that returns a line of text as typed by the user. There are two available constructors for creating a Scanner object. For console input, it requires only one argument, an instance of an InputStream object.

15. Write the description of Java.io.FileInputStream.getFD() Method.

Java.io.FileInputStream.getFD() Method returns the object of FileDescriptor that identifies the connection to the actual file in the file system being used by this FileInputStream.

5 MARK

1.Write notes on Languages in Java programming.

Boolean class --- Encapsulates a Boolean value and the constants are Flase and True.Boolean,Boolean Value,getboolean

Byte class ------ Encapsulates a Byte value and two byte constants are Min-value and Max-value.Byte,bytevalue,decode,doublevalue.

Character class ----- Encapsulates a Character value and two int constants are Max-radix and Min-radix.Character,charvalue,digit,hashcode.

The Class Class ---------Encapsulates the rub-time state of an object.forName,getinterface,getname,getsuperclass.isinterface.

2.Write notes on Thread.

Thread class encapsulates a thread of execution and provides several methods help manages threads.
Public thread(),publicThread(runnablethreadOb),publicThread(runnable threadOb,string threadName)

3.Explain the utility classes in java.

Classes found in java.util as bitset,calendar,date,dictionary,eventobject,locale,hash table. Defines Enumeration,Evenlistener,observer.

4.Briefly explain the stringtokenizer class.

Stringtokenizer class used to break a string into its indidual tokens.
Nexttoken method(),hasmoretokens()method,hasmoreelements() nextelement() methods.counttokens- public int counttokens()

5.Explain the bufferedinputstream class in brief.

BufferedInputStream class allows to wrap any InputStream into a buttfered stream and achieve a performance improvement. Classes are FilterInpputStream,InputStream

10 MARK

1. Explain in detail about the String Class in java programming to manipulate the string.

String - Provide a set of methods to manipulate a string.Classes are StringBuffer. Public string(),public string(byte asciichars[]),public string(byte asciichars[],int highorderbyte)public string(byte asciichars[],int start,int size)

2.Discuss the EventListener interface in java specifications to identify the object.

EventListener interface is extended by all event listener interfaces. Contains no methoids but used to identify the object whether it listens the events.Eventobject class,eventobject,getsource,to string

3.Explain the file class to manipulate the information with disk file or directory.

File does not operate on streams.Most common methods are FileDescriptor,File,canRead,canWrite,delete,equals,exists,getAbsolutepath,getname,getparent,getpath,hashcode

UNIT IV
2 MARK

1.What is the networking classes for web-based programming?

Java.net contains classes as
contenthandler,inetaddress,URL,datagrampacket,datagramsocket
serversocket,URLconnection,URLencoder, URLstreamhandler.

2.What URLEncoder class?

URLEncoder class contains static method that takes a String object and converts to corresponding URL-encoded.

3.Mention about InetAddress class.

InetAddress class provides methods for working with internet addresses.

4.Define AppletClass.

Applet class contains several methods to control over the execution of applet.All applets are subclasses of Applet.

5.How the destroy the execution environment before an applet is terminated.

Using public void destroy() method.The execution environment is terminated before an applet.

6.What is the use of Audioclip interface?

AudioClip is an interface to get and control audio files.

7.Mention the different classes of Applet methods.

Component,Container,panel are the classes of Applet methods.

8.What is SocketClass?

Socket Class is designed to connect to server sockets and initiate protocol exchanges.

9.What is the setSoTimeout?

Sets the timeout period for invoking serversocket.Determines how long accept() waits for a connection request.

10. *Definition of socket.*

A socket is one end-point of a two-way communication link between two programs running on the network.

11. What are the networking classes in JDK?

The classes in `java.net`, Java programsTCP or UDP used to communicate over the Internet. The `URL`, `URLConnection`, `Socket`, and `ServerSocket` classes all use TCP to communicate over the network. The `DatagramPacket`, `DatagramSocket`, and `MulticastSocket` classes are for use with UDP.

12. How to create URL for web programming?

To create a URL object is from a String that represents the human-readable form of the URL address. Another person can use a URL.Java program, using a String containing the text to create a URL object:

URL myURL = new URL("http://example.com/");

The URL object created above represents an *absolute URL*. An absolute URL contains all of the information necessary to reach the resource in question.Also URL objects from a *relative URL* address can be created.

13.What is Bind?

public void bind(SocketAddress addr)
throws SocketException

Binds this DatagramSocket to a specific address & port.

14. What is SocketException?

SocketException - if any error happens during the bind, or if the socket is already bound.

15.What is role of void play,void loop and void stop in java networking classes?

void play () :The play() method plays the audio clip once from the beginning.
void loop ():The loop() method plays the audio clip continuously. When it gets to the end-of-file marker, it resets itself to the beginning.
void stop ():The stop() method stops the applet from playing the audio clip.

5 MARK

1.Briefly write note on TCP and UDP.

TCP

TCP provides a point-to-point channel for applications that require reliable communications. The Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), and Telnet are all examples of applications that require a reliable communication channel. *TCP* (*Transmission Control Protocol*) is a connection-based protocol that provides a reliable flow of data between two computers.

UDP

The UDP protocol provides for communication that is not guaranteed between two applications on the network. UDP is not connection-based like TCP. Rather, it sends independent packets of data, called *datagrams*, from one application to another. *UDP* (*User Datagram Protocol*) is a protocol that sends independent packets of data, called datagrams, from one computer to another with no guarantees about arrival. UDP is not connection-based like TCP.

2.Explain how to read from and write to a socket.

1. Open a socket.
2. Open an input stream and output stream to the socket.
3. Read from and write to the stream according to the server's protocol.
4. Close the streams.
5. Close the socket.

3.How to Broadcast to Multiple Recipients?Explain.

It broadcasts `DatagramPacket`s to multiple recipients. Instead of sending quotes to a specific client that makes a request, the new server now needs to broadcast quotes at a regular interval. The client needs to be modified so that it passively listens for quotes and does so on a `MulticastSocket`.

Three classes which are modifications of the three classes from the previous example: `MulticastServer`, `MulticastServerThread`, and `MulticastClient`. This discussion highlights the interesting parts of these classes.

4.What are the life cycle of an applet in brief?Explain.

Life Cycle of an Applet:

Four methods in the Applet class give you the framework on which you build any serious applet:

- init: This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed.
- start: This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.
- stop: This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.
- destroy: This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.
- paint: Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.

5. Write notes on Event Handling for Applets.

Applets inherit a group of event-handling methods from the Container class. The Container class defines several methods, such as processKeyEvent and processMouseEvent, for handling particular types of events, and then one catch-all method called processEvent.

The applet displays "initializing the applet. Starting the applet" click inside the rectangle "mouse clicked" be displayed as well.

10 MARK

1. Explain *Java Network Programming in detail.*

`ObjectOutputStream` inherits from `OutputStream` and implements the `ObjectOutput` interface. `ObjectOutput` then implements the `DataOutput` interface. `ObjectOutputStream.writeInt()` is inherited from `DataOutputStream.writeInt()`, implying that `ObjectOutputStream` streams are somehow compatible with `DataOutputStream` streams, which is again false. The streams produced by `ObjectOutputStream` and `DataOutputStream` are totally incompatible, which is why JavaSoft couldn't subclass `DataOutputStream` to produce `ObjectOutputStream` in the first place.

2.Explain about Encapsulation,Language Mapping,Serialization,Thread and UDP in detail.

Encapsulation: Object-oriented programming, practice of.
Language Mapping: The means necessary to take one language and convert its syntax and semantics to another language.
Serialization: The act of transforming a Java object into a string representation.
Thread: A series of executable steps that are executed along with other steps.

UDP: Unreliable Datagram Protocol.

3.Explain the java applet in detail for java programming.

java.applet

Class Applet

java.lang.Object

java.awt.Component

java.awt.Container

java.awt.Panel

java.applet.Applet

4.Briefly explain the URL class which provides accessor methods in Java.

getProtocol
Returns the protocol identifier component of the URL.
getAuthority
Returns the authority component of the URL.
getHost

Returns the host name component of the URL.

getPort

Returns the port number component of the URL. The getPort method returns an integer that is the port number. If the port is not set, getPort returns -1.

getPath

Returns the path component of this URL.

getQuery

Returns the query component of this URL.

getFile

Returns the filename component of the URL. The getFile method returns the same as getPath, plus the concatenation of the value of getQuery, if any.

getRef

Returns the reference component of the URL.

UNIT V
2 MARK

1.What is Event classes?

Event classes represent the event. Java provides us various Event classes but we will discuss those which are more frequently used.

2.How to declare the class for Eventobject class in java?

Following is the declaration for java.util.EventObject class:

1.public class EventObject
2.extends Object
3.implements Serializable

3.Mention the Class constructors to represent in Menu class of Java programming.

1.Menu() : Constructs a new menu with an empty label.

2. Menu(String label): Constructs a new menu with the specified label.

3.Menu(String label, boolean tearOff) :

Constructs a new menu with the specified label, indicating whether the menu can be torn off.

4.How the class is declared in Window Class in java?

Following is the declaration for java.awt.Window class:

1.public class Window
2.extends Containe
3.implements Accessible

5.Mention the class constructors in FrameClass of AWT in java programming.

Frame() :Constructs a new instance of Frame that is initially invisible.

Frame(GraphicsConfiguration gc) :Constructs a new, initially invisible Frame with the specified GraphicsConfiguration.

Frame(String title) : Constructs a new, initially invisible Frame object with the specified title.

Frame(String title, GraphicsConfiguration gc): Constructs a new, initially invisible

Frame object with the specified title and a GraphicsConfiguration.

6.Why AWT ActionListener Interface is implemented in java?

The class which processes the ActionEvent should implement this interface.The object of that class must be registered with a component. The object can be registered using the addActionListener() method. When the action event occurs, that object's actionPerformed method is invoked.

7.Why low-level-event is generated in MouseEvent Class for Event Classes?

Low-level event is generated by a component object for Mouse Events and Mouse motion events.

- a mouse button is pressed
- a mouse button is released
- a mouse button is clicked (pressed and released)
- a mouse cursor enters the unobscured part of component's geometry
- a mouse cursor exits the unobscured part of component's geometry
- a mouse is moved
- the mouse is dragged

8.What is the necessary of AdjustmentListener for receiving adjustment events?

InterfaceAdjustmentListener is used for receiving adjustment events. The class that process adjustment events needs to implements this interface.

9.What are the fields of FileDialog Class?

1.static int LOAD -- This constant value indicates that the purpose of the file dialog window is to locate a file from which to read.

2.static int SAVE -- This constant value indicates that the purpose of the file dialog window is to locate a file to which to write.

10.What are the different Class constructors in Button Class?

1.Button() :Constructs a button with an empty string for its label.

2. Button(String text) :Constructs a new button with specified label.

11.What Panel class?

The class Panel is the simplest container class. It provides space in which an application can attach any other component, including other panels. It uses FlowLayout as default layout manager.

12.What is Toolkit Class in window class?

Toolkit is an abstract class,defines methods used in the process of creating platform-dependent for various GUI components.

13.Define the ArrayList class of List Class.

The java.util.ArrayList class provides resizable-array and implements the List interface.Following are the important points about ArrayList:

- It implements all optional list operations and it also permits all elements, includes null.
- It provides methods to manipulate the size of the array that is used internally to store the list.
- The constant factor is low compared to that for the LinkedList implementation.

14.How interface is similar to a class in java?

- An interface can contain any number of methods.
- An interface is written in a file with a .java extension, with the name of the interface matching the name of the file.
- The bytecode of an interface appears in a .class file.
- Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

15.Define AWT TextArea Class in java

The TextArea control in AWT provide us multiline editor area. The user can type here as much as he wants. When the text in the text area become larger than the viewable area the scroll bar is automatically appears which help us to scroll the text up & down and right & left.

5 MARK

1.Brifely write about AWTEvent Classes utilized in AWT Event class.

AWTEvent:
It is the root event class for all AWT events. This class and its subclasses supercede the original java.awt.Event class.

ActionEvent:
The ActionEvent is generated when button is clicked or the item of a list is double clicked.

InputEvent:
The InputEvent class is root event class for all component-level input events.

KeyEvent:
On entering the character the Key event is generated.

MouseEvent:
This event indicates a mouse action occurred in a component.

TextEvent:
The object of this class represents the text events.

WindowEvent:
The object of this class represents the change in state of a window.

AdjustmentEvent:
The object of this class represents the adjustment event emitted by Adjustable objects.

ComponentEvent:
The object of this class represents the change in state of a window.

ContainerEvent:
The object of this class represents the change in state of a window.

MouseMotionEvent:
The object of this class represents the change in state of a window.

PaintEvent:
The object of this class represents the change in state of a window.

2.Briefly write the notes on Component Class in java.

Class ComponentEvent represents that a component moved, changed size, or changed visibility

Class declaration

Declaration for java.awt.event.ComponentEvent class:

public class ComponentEvent
  extends AWTEvent

Field

Fields for java.awt.Component class:

- static int COMPONENT_FIRST -- The first number in the range of ids used for component events.
- static int COMPONENT_HIDDEN --This event indicates that the component was rendered invisible.
- static int COMPONENT_LAST -- The last number in the range of ids used for component events.
- static int COMPONENT_MOVED -- This event indicates that the component's position changed.
- static int COMPONENT_RESIZED -- This event indicates that the component's size changed.
- static int COMPONENT_SHOWN -- This event indicates that the component was made visible.

Class constructors

Constructor & Description

ComponentEvent(Component source, int id)
Constructs a ComponentEvent object.

Class methods

Method & Description

Component getComponent()
Returns the originator of the event.

String paramString()
Returns a parameter string identifying this event.

Methods inherited

Interface inherits methods from the following classes:

- java.awt.AWTEvent
- java.util.EventObject
- java.lang.Object

3.Write notes on Dialog Class in java programming.

Dialog control represents a top-level window with a title and a border used to take some form of input from the user.

Class declaration

java.awt.Dialog class:

public class Dialog
extends Window

the fields for java.awt.Image class:

- static Dialog.ModalityType DEFAULT_MODALITY_TYPE -- Default modality type for modal dialogs.

Class constructors

Class methods

Methods inherited

This class inherits methods from the following classes:

- java.awt.Window
- java.awt.Component
- java.lang.Object

4.How Graphics class is abstract super class of graphics contexts in java?

The Graphics class is the abstract super class for all graphics contexts which allow an application to draw onto components that can be realized on various devices, or onto off-screen images as well.

A Graphics object encapsulates all state information required for the basic rendering operations that Java supports. State information includes the following properties.

- The Component object on which to draw.
- A translation origin for rendering and clipping coordinates.
- The current clip.
- The current color.
- The current font.
- The current logical pixel operation function.
- The current XOR alternation color

5.Explain the KeyEvent Class in AWT of java.

On entering the character the Key event is generated.There are three types of key events which are represented by the integer constants. These key events are following

- KEY_PRESSED
- KEY_RELASED
- KEY_TYPED

10 MARK

1.Discuss the AWT checkbox class constructors and methods in java.

Checkbox control is used to turn an option on(true) or off(false). There is label for each checkbox representing what the checkbox does.The state of a checkbox can be changed by clicking on it.

Class declaration:

Following is the declaration for java.awt.Checkbox class:

public class Checkbox
extends Component
implements ItemSelectable,Accessible

Class constructors:

Checkbox() :
Creates a check box with an empty string for its label.

Checkbox(String label) :
Creates a check box with the specified label.

Checkbox(String label, boolean state) :
Creates a check box with the specified label and sets the specified state.

Checkbox(String label, boolean state, CheckboxGroup group) :
Constructs a Checkbox with the specified label, set to the specified state, and in the specified check box group.

Checkbox(String label, CheckboxGroup group, boolean state) :
Creates a check box with the specified label, in the specified check box group, and set to the specified state.

Methods inherited:

This class inherits methods from the following classes:

- java.awt.Component
- java.lang.Object
-

2.Explain the AWT Layouts in java to arrange the components within the container.

Layout means the arrangement of components within the container. In other way we can say that placing the components at a particular position within the container. The task of layouting the controls is done automatically by the Layout Manager.

Layout Manager

The layout manager automatically positions all the components within the container. If we do not use layout manager then also the components are positioned by the default layout manager. It is possible to layout the controls by hand but it becomes very difficult because of the following two reasons.

- It is very tedious to handle a large number of controls within the container.
- Oftenly the width and height information of a component is not given when we need to arrange them.

Java provide us with various layout manager to position the controls. The properties like size,shape and arrangement varies from one layout manager to other layout manager. When the size of the applet or the application window changes the size, shape and arrangement of the components also changes in response i.e. the layout managers adapt to the dimensions of appletviewer or the application window.

The layout manager is associated with every Container object. Each layout manager is an object of the class that implements the LayoutManager interface.

Functionalities of Layout Managers:

LayoutManager:
The LayoutManager interface declares those methods which need to be implemented by the class whose object will act as a layout manager.

LayoutManager2:
The LayoutManager2 is the sub-interface of the LayoutManager.This interface is for those classes that know how to layout containers based on layout constraint object.

AWT Layout Manager Classes:

Following is the list of commonly used controls while designed GUI using AWT.

BorderLayout:
The borderlayout arranges the components to fit in the five regions: east, west, north, south and center.

CardLayout:
The CardLayout object treats each component in the container as a card. Only one card is visible at a time.

FlowLayout:
The FlowLayout is the default layout.It layouts the components in a directional flow.

GridLayout:
The GridLayout manages the components in form of a rectangular grid.

GridBagLayout:
This is the most flexible layout manager class.The object of GridBagLayout aligns the component vertically,horizontally or along their baseline without requiring the components of same size.

3.Discuss the WindowEvent to implement AWT WindowListener Interface in detail.

The class which processes the WindowEvent should implement this interface.The object of that class must be registered with a component. The object can be registered using the addWindowListener() method.

Interface declaration:

java.awt.event.WindowListener interface:

public interface WindowListener
extends EventListener

Methods inherited

Inherits methods from the following interfaces:

- java.awt.EventListener