Notes:

->Java object code is stored in files with the extension .class.

->Java programs are executed by the Java interpreter, which analyses the bytecodes and carries out the operations they specify.

->The Java Development Kit (JDK) supports the compilation and execution of Java applications and applets.

->This is because == operator compares memory locations, while equals() method compares the content stored in two objects.

->Automatic Type Conversions in Assignments

byte → short → int → long → float → double

Q1. Short Note on Documentation Comments.

Ans.

->You can also include comments in a program that are intended to produce separate documentation for the program. These are called Documentation Comments.

->A program called javadoc processes the documentation comments in the source code for a program to generate separate documentation for the code.

->The documentation that is generated by javadoc is in the form of HTML web pages that can be viewed using a browser such as Internet Explorer.

->A documentation comment begins with /** and ends with */.

->e.g. /** This is a documentation comment. */

->Any asterisks at the beginning of each line in a documentation comment are ignored, as are any spaces preceding the first asterisk.

->A documentation comment also include HTML tags except for header tags, as well as special tags beginning with @. The @ character is followed by a keyword that defines the purpose of the tag.

->e.g.

```
/** @author Ivor Horton */

/** @exception IOException When an I/O error occurs. */
```

Q2. Why string are immutable?

Ans.

->The contents of a String object are immutable. That is, once created, the character sequence that makes up the string cannot be altered.

->The key benefits of keeping String class as immutable are caching, security, synchronization, and performance.

->This restriction allows Java to implement strings more efficiently. Even though this probably sounds like a serious drawback, it is not.

->When you need a string that is a variation on one that already exists, simply create a new string that contains the desired changes.

->Since unused String objects are automatically garbage collected, you don't even need to worry about what happens to the discarded strings.

->String reference variables may, of course, change the object to which they refer. It is just that the contents of a specific String object cannot be changed after it is created.

->Java also offers a class called StringBuffer, which creates string objects that can be changed.

For example, in addition to the charAt( ) method, which obtains the character at a specific location, StringBuffer defines setCharAt( ), which sets a character within the string.

->Java also supplies StringBuilder, which is related to StringBuffer, and also supports strings that can be changed.

example.

```
class Test{
 public static void main(String args[]){
   String s="Sachin";
   s.concat(" Tendulkar"); //concat() method appends the string at the end
   System.out.println(s); //will print Sachin because strings are immutable objects
 }
}
```

O/P: Sachin

Q3. Differentiate between == and equals().

Ans.

Sr. No.	Key	==	equals() method
1	Type	== is an operator.	equals() is a method of Object class.
2	Comparision	== should be used during reference comparison. == checks if both references points to same location or not.	equals() method should be used for content comparison. equals() method evaluates the content to check the equality.
2	Object	== operator can not be overriden.	equals() method if not present and Object.equals() method is utilized, otherwise it can be overridden.

```
public class JavaTester {
   public static void main(String args[]) {
      String s1 = new String("TUTORIALSPOINT");
      String s2 = new String("TUTORIALSPOINT");
      //Reference comparison
      System.out.println(s1 == s2);
      //Content comparison
      System.out.println(s1.equals(s2));
      // integer-type
      System.out.println(10 == 10);
      // char-type
      System.out.println('a' == 'a');
   }
}
```

Output

```
false
true
true
true
```

Q4. The universal super class/object class.

Ans.

->All classes have a standard class, Object, as a base, so Object is a superclass of every class, that is Universal superclass.

->No need to specify the class Object as a base in the definition of classes — it happens automatically.

->A variable of type Object can store a reference to an object of any class type. This is useful when you want to write a method that needs to handle objects of unknown type.

->You can define a parameter to the method of type Object, in which case a reference to any type of object can be passed to the method.

->Classes will inherit members from the class Object.

Q5. Instance Method and Instance Variable and Class Methods in java.

Ans.

Instance Method:

-> If we do not uses the static keyword with variable/method than it belongs or categorized as instance method which is defined at instance level and need class object for their accessibility.

-> Also static methods exist as a single copy for a class while instance methods exist as multiple copies depending on the number of instances created for that particular class.

-> Static methods can't access instance methods/variables directly while instance methods can access static variables and static methods directly.

Example.

```
public int getnumber(){

            return num;

}
```

Instance Variable:

->Instance variables are variables within a class but outside any method.

->These variables are initialized when the class is instantiated.

->Instance variables can be accessed from inside any method, constructor or blocks of that particular class.

Example.

```
class helloworld{

      int num;      //instance variable;

     static int num3;  //class variable

}
```

Class Methods:

-> A methods are declared within a class, and that they are used to perform certain actions is called class methods.

Example.

```
public class MyClass {
  static void myMethod() {
    System.out.println("Hello World!");
  }
}
```

->Creating a method named myMethod() in MyClass. myMethod() will print the corresponding string when called.

Q6. Discuss the use of the static keyword to create fields belong to the class.

Ans.

->In Java, static keyword is mainly used for memory management.

->It can be used with variables, methods, blocks and nested classes.

->It is a keyword which is used to share the same variable or method of a given class.

->Basically, static is used for a constant variable or a method that is same for every instance of a class.

->The main method of a class is generally labeled static.

Q7. Explain class variable/static variables and static methods in Java.

Ans.

Static Variable:

->When a variable is declared as static, then a single copy of variable is created and shared among all objects at class level.

->Static variables are, essentially, global variables. All instances of the class share the same static variable.

Example.

```
class helloworld{
     int num;     //instance variable;
    static int num3;  //class variable or static variable
}
public static void main(){
        helloworld obj = new helloworld();
        obj.num =10;        //instance variable call
        helloworld.num2 = 20;   //class variable call
}
```

Static Method:

->Static method is a method that belongs to a class rather than an instance of a class.

->The method is accessible to every instance of a class, but methods defined in an instance are only able to be accessed by that member of a class.

->Static methods do not use any instance variables of any object of the class they are defined in.

->Static methods take all the data from parameters and compute something from those parameters, with no reference to variables.

->Class variables and methods can be accessed using the class name followed by a dot and the name of the variable or method.

->When a method is declared with the static keyword, it is known as a static method.

->The most common example of a static method is the main( ) method.

->Methods declared as static can have the following restrictions:

- They can directly call other static methods only.
- They can access static data directly.

Q8. Explain Abstract Class and Abstract Method in java.

Ans.

->Abstraction is nothing but a process of hiding the implementation details and showing only functionality to the user.

->Abstraction lets you focus on what the object does instead of how it does it.

Abstract Class:

->A class which is declared with the abstract keyword is known as an abstract class in Java.

->It can have abstract and non-abstract methods (method with the body).

->It needs to be extended and its method implemented. It cannot be instantiated.

Rules for Abstract Class:

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

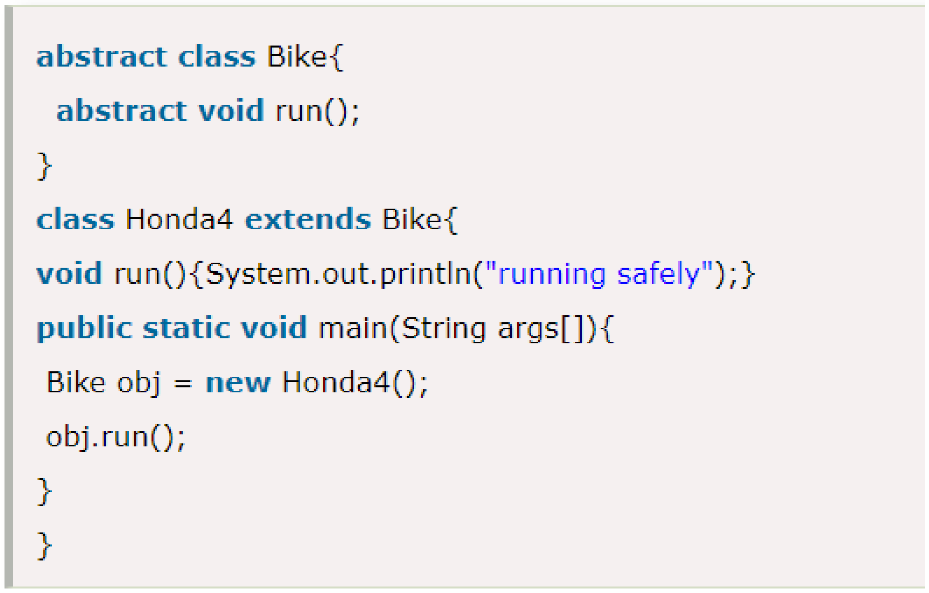
Abstract Method:

->A method which is declared as abstract and does not have implementation is known as an abstract method.

Syntax:

```
abstract void printStatus(); //no method body and abstract
```

Example.



```
abstract class Bike{
  abstract void run();
}
class Honda4 extends Bike{
void run(){System.out.println("running safely");}
public static void main(String args[]){
 Bike obj = new Honda4();
 obj.run();
}
}
```

->In this example, Bike is an abstract class that contains only one abstract method run. Its implementation is provided by the Honda class.

Q9. Why should we have util package?

Ans.

->Java util package contains the collections framework, collection of classes and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array).

Important Classes in Java.util.package:

1. Arrays: This class contains various methods for manipulating arrays (such as sorting and searching).
2. Scanner: A simple text scanner which can parse primitive types and strings using regular expressions.
3. Stack: The Stack class represents a last-in-first-out (LIFO) stack of objects.
4. StringTokenizer: The string tokenizer class allows an application to break a string into tokens.
5. Random: An instance of this class is used to generate a stream of pseudorandom numbers.

Q10. Compare autoboxing and unboxing in Java.

Ans.

Autoboxing:

-> Converting a primitive value into an object of the corresponding wrapper class is called autoboxing.

->For example, converting int to Integer class. The Java compiler applies autoboxing when a primitive value is:

- Passed as a parameter to a method that **expects an object** of the corresponding wrapper class.
- Assigned to a variable of the corresponding **wrapper class**.

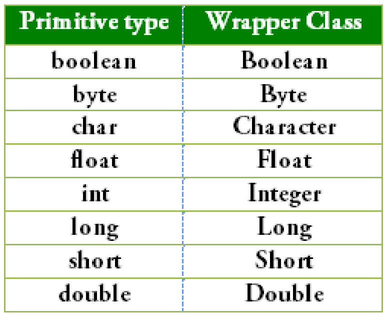
Unboxing:

->Converting an object of a wrapper type to its corresponding primitive value is called unboxing.

->For example conversion of Integer to int. The Java compiler applies unboxing when an object of a wrapper class is:

- Passed as a parameter to a method that **expects a value** of the corresponding primitive type.
- Assigned to a variable of the corresponding **primitive type**.

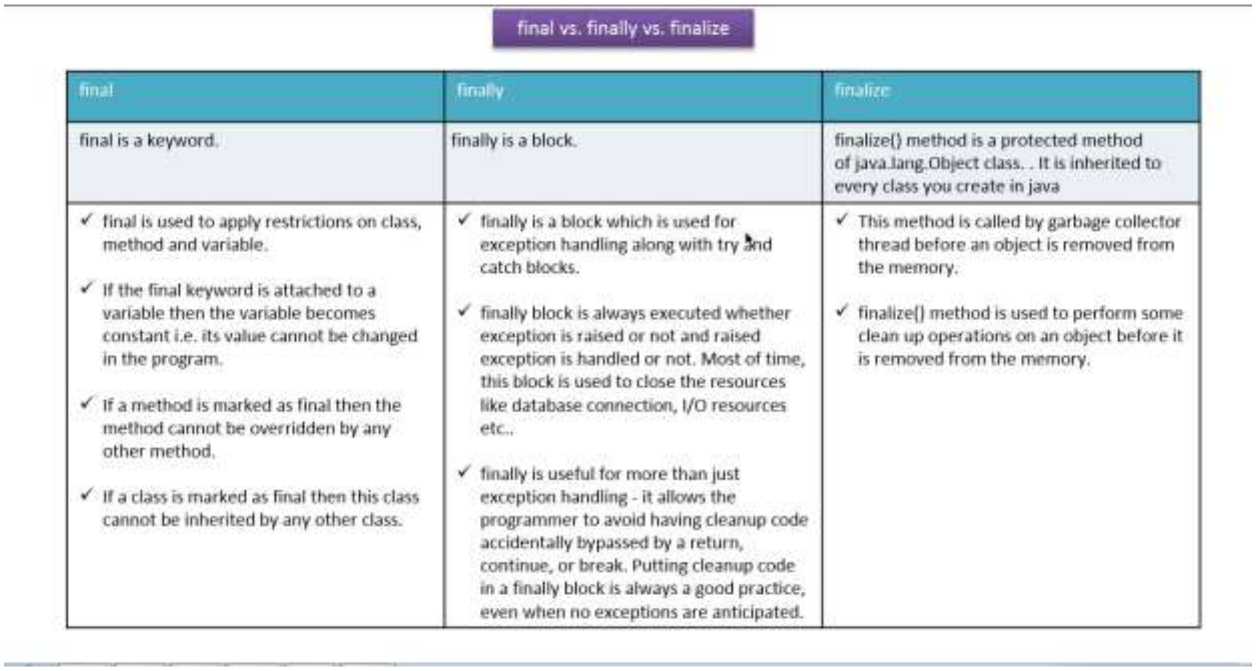
The following table lists the primitive types and their corresponding wrapper classes, which are used by the Java compiler for autoboxing and unboxing:



Primitive type	Wrapper Class
boolean	Boolean
byte	Byte
char	Character
float	Float
int	Integer
long	Long
short	Short
double	Double

Q11. Difference between finally, finalize and final keyword.

Ans.



final vs. finally vs. finalize

final	finally	finalize
final is a keyword.	finally is a block.	finalize() method is a protected method of java.lang.Object class. . It is inherited to every class you create in java
✓ final is used to apply restrictions on class, method and variable. ✓ If the final keyword is attached to a variable then the variable becomes constant i.e. its value cannot be changed in the program. ✓ If a method is marked as final then the method cannot be overridden by any other method. ✓ If a class is marked as final then this class cannot be inherited by any other class.	✓ finally is a block which is used for exception handling along with try and catch blocks. ✓ finally block is always executed whether exception is raised or not and raised exception is handled or not. Most of time, this block is used to close the resources like database connection, I/O resources etc.. ✓ finally is useful for more than just exception handling - it allows the programmer to avoid having cleanup code accidentally bypassed by a return, continue, or break. Putting cleanup code in a finally block is always a good practice, even when no exceptions are anticipated.	✓ This method is called by garbage collector thread before an object is removed from the memory. ✓ finalize() method is used to perform some clean up operations on an object before it is removed from the memory.

Example for final.

```
class FinalExample{

public static void main(String[] args){
        final int x=100;
                x=200; //Compile Time Error
        }
}
```

Example for finally.

```
class FinallyExample{
        public static void main(String[] args){
                try{
                        int x=300;
                }catch(Exception e) {System.out.println(e);}
                Finally
                {
                System.out.println("finally block is executed");
                }
        }
}
```

Example for finalize.

```
class FinalizeExample{
        public void finalize(){System.out.println("finalize called");}
                public static void main(String[] args){
                        FinalizeExample f1=new FinalizeExample();
                        FinalizeExample f2=new FinalizeExample();
                f1=null;
                f2=null;
        System.gc();
}
}
```

Q12. Discuss difference between while loop and do while loop.

Ans.

while loop:

->while loop evaluates the condition first and then execute the statements.

->the condition is specified at the beginning of the loop.

->the body is executed only if a certain condition is met and it terminates when the condition is false.

->while loop condition did not contain semicolon(;).

->syntax: while(condition)

```
        {
        Statements
        }
```

do while:

->do while loop execute the statements first before evaluating the condition.

->the condition isn't specified until after the body of the loop.

->the body is always executed at least once, regardless of whether the condition is met.

->do while loop condition contain semicolon(;).

->syntax: Do

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

Q13. Discuss Wrapper classes with example or Explain Wrapper class.

Ans.

-> Wrapper classes are those whose objects wraps a primitive data type within them.

->In the java.lang package java provides a separate class for each of the primitive data types namely Byte, Character, Double, Integer, Float, Long, Short.

->At the time of instantiation, these classes accept a primitive datatype directly, or in the form of String.

->Wrapper classes provide methods to, convert primitive datatypes within them to String objects and, to compare them with other objects etc.

->Using wrapper classes, you can also add primitive datatypes to various Collection objects such as ArrayList, HashMap etc. You can also pass primitive values over a network using wrapper classes.

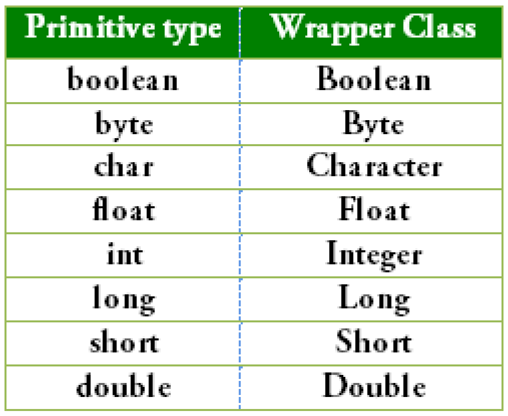
->example:

```
public class MyClass {

  public static void main(String[] args) {

    Integer myInt = 5;

    Double myDouble = 5.99;

    Character myChar = 'A';

    System.out.println(myInt.intValue());

    System.out.println(myDouble.doubleValue());

    System.out.println(myChar.charValue());

  }

}
```

->Primitive Data Type	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character

->In addition to allowing a primitive type to be passed by reference, these wrapper classes define several methods that enable you to manipulate their values.

->For example, the numeric type wrappers include methods that convert a numeric value from its binary form into its human-readable String form, and vice versa.



Primitive type	Wrapper Class
boolean	Boolean
byte	Byte
char	Character
float	Float
int	Integer
long	Long
short	Short
double	Double

Q14. Discuss Queue.

Ans:

->A queue is a list in which elements can be accessed in first-in, first-out (FIFO) order only.

->A queue is like a line at a bank—the first in line is the first served.

->In general, queues support two basic operations: put and get. t. Each put operation places a new element on the end of the queue. Each get operation retrieves the next element from the front of the queue

->Queue operations are consumptive: once an element has been retrieved, it cannot be retrieved again. The queue can also become full, if there is no space available to store an item, and it can become empty, if all of the elements have been removed.

->There are two basic types of queues—circular and noncircular. A circular queue reuses locations in the underlying array when elements are removed. A noncircular queue does not reuse locations and eventually becomes exhausted.

Q15. What is Garbage collector.

Ans:

->Garbage collection looking at heap memory, identifying which objects are in use and which are not, and deleting the unused objects.

->An in use object, or a referenced object, means that some part of your program still maintains a pointer to that object. An unused object, or unreferenced object, is no longer referenced by any part of your program. So the memory used by an unreferenced object can be reclaimed.

->A garbage collector performs automatic dynamic memory management through the following operations:

i)Allocates from and gives back memory to the operating system.

ii)Hands out that memory to the application as it requests it.

iii)Determines which parts of that memory is still in use by the application.

iv)Reclaims the unused memory for reuse by the application.

->It is automatically done by the garbage collector(a part of JVM).

->Example:

```
Integer i = new Integer(4);

// the new Integer object is reachable  via the reference in 'i'

i = null;

// the Integer object is no longer reachable.
```

->Here after i = null; integer object 4 in heap area is eligible for garbage collection. And the Memory is free by garbage collector.

Q16. Explain Interface.

Ans:

->The interface in Java is a mechanism to achieve abstraction

->Interface looks like a class but it is not a class. An interface can have methods and variables just like the class but the methods declared in interface are by default abstract (only method signature, no body).

->Also, It has static constants and abstract methods and the variables declared in an interface are public, static & final by default.

->By interface, we can support the functionality of multiple inheritance.

-> Since methods in interfaces do not have body, they have to be implemented by the class before you can access them. The class that implements interface must implement all the methods of that interface.

->Also, java programming language does not allow you to extend more than one class, However you can implement more than one interfaces in your class.

->Syntax:

Interfaces are declared by specifying a keyword "interface".

```
interface MyInterface

{

  /* All the methods are public abstract by default

   * As you see they have no body

   */

  public void method1();

  public void method2();

}
```

```
->class---extends--->class

->interface---extends--->interface

->class---implements--->interface1

   |

   |------implements--->interface2
```

Q17. Compare static fields Vs non static fields.

Ans:

Static fields:

i. Static variables can be accessed using class name

ii. Static variables can be accessed by static and non static methods

iii. Static variables reduce the amount of memory used by a program.

iv. Static variables are shared among all instances of a class.

v. Static variable is like a global variable and is available to all methods.

Non static fields:

i. Non static variables can be accessed using instance of a class.

ii. Non static variables cannot be accessed inside a static method.

iii.Non static variables do not reduce the amount of memory used by a program.

iv. Non static variables are specific to that instance of a class.

v.Non static variable is like a local variable and they can be accessed through only instance of a class.

Q18. Compare class variables vs instance variables.

Ans:

Class Variable:

i. Class variables are declared with keyword static.

ii. Class variables are common to all instances of a class. These variables are shared between the objects of a class.

iii. As class variables are common to all objects of a class, changes made to these variables through one object will reflect in another.

iv. Class variables can be accessed using either class name or object reference.

Instance Variable:

i. Instance variables are declared without static keyword.

ii. Instance variables are not shared between the objects of a class. Each instance will have their own copy of instance variables.

iii. As each object will have its own copy of instance variables, changes made to these variables through one object will not reflect in another object.

iv. Instance variables can be accessed only through object reference.

Q19. Difference between static and final.

Ans:

Static:

i. static means there is only one copy of the variable in memory shared by all instances of the class.

ii. It is not compulsory to initialize the static variable at the time of its declaration.

iii. The static variable can be reinitialized

iv. Static methods can only access the static members of the class, and can only be called by other static methods.

v. Static class's object can not be created, and it only contains static members only.

vi. Static block is used to initialize the static variables.

Final:

i. The final keyword just means the value can't be changed. Without final, any object can change the value of the variable.

ii. It is compulsory to initialize the final variable at the time of its declaration.

iii. The final variable can not be reinitialized.

iv. Final methods can not be inherited.

v. A final class can not be inherited by any class.

vi. Final keyword supports no such block.

Q20. Difference between primitive and referenced type

Ans:

Primitive:

i. primitive type is used for storing simple values ->byte,short,int,long,float,double,char,boolean

ii. when declaring primitive type we don't need to allocate memory. Memory is allocated and relesed by java runtime environment (JRE) eg:- byte age=30;

iii. primitive types don't have members.Eg:- we cant write age.getAge()

iv. primitive types are copied by the value. Primitive - x and y both have diffrent memory loactions so it is not affected.

eg:-

```
byte x=1;

byte y=x;

x=2;                    //if we change the x value

System.out.println(y);    // 1
```

v. primitive type always has a value, it can never be null

vi. primitive types store actual values but reference type stores handle to object in the heap

vii. To create a variable of primitive type 'new' keyword is not used

Referenced :

i. referenced type is used for storing complex values -> Date, String

ii. when dealing with referenced type we should always allocate memory and we don't need to related the memory JRE take care of it

eg:-Date now = new Date();

iii. Referenced type have members

eg:-

```
String myStr = "Hello";

char result = myStr.charAt(0);
```

iv. referenced types are copied by the references

eg:-

```
int x=1,y=1;

Point point1 = new Point(x,y);

Point point2 = point1; //it store the address of point1

point1.x = 2;


System.out.println(point1);                    //java.awt.Point[x=2,y=1]
System.out.println(point2);                    //java.awt.Point[x=2,y=1]
```

v. reference type can be null, which denotes the absence of value.

vi. Reference variables are not pointers but a handle to the object which is created in heap memory.

vii.To create a variable of reference type 'new' keyword is used.

Q21. Explain Constructors and its type.

Ans:

->A constructor in Java is a special method that is used to initialize objects.

->When you create an object of a class, a special kind of method called a constructor is always invoked. If you don’t define any constructors for your class, the compiler will supply a default constructor in the class, which does nothing.

->The default constructor is also described as the no-argument constructor because it requires no arguments to be specified when it is called.

->The primary purpose of a constructor is to provide you with the means of initializing the instance variables uniquely for the object that is being created.

Rules to define constructor:-

->Constructor(s) of a class must have same name as the class name in which it resides.

->A constructor in Java can not be abstract, final, static and Synchronized.

->Access modifiers can be used in constructor declaration to control its access i.e which other class can call the constructor.

constructors are different from methods in Java:-

->Constructor(s) must have the same name as the class within which it defined while it is not necessary for the method in java.

->Constructor(s) do not return any type while method(s) have the return type or void if does not return any value.

->Constructor is called only once at the time of Object creation while method(s) can be called any numbers of time.

Types of constructor:-

1) No argument Constructors/Default Constructor

->constructor does not accept any parameter

```
public class MyClass {
  int x;
  // Create a class constructor for the MyClass class
  public MyClass() {
    x = 5;
  }
  public static void main(String[] args) {
    MyClass myObj = new MyClass();
    System.out.println(myObj.x);  //5
  }
}
```

2)Parameterized Constructors

->constructor that accepts one or more parameters.

```
public class MyClass {
  int x;
  // Create a class constructor for the MyClass class
  public MyClass(int i) {
    x = i;
  }
  public static void main(String[] args) {
    MyClass myObj = new MyClass(10);
    System.out.println(myObj.x);                //10
  }
}
```

Q22. Explain Finally.

Ans:

->Java finally block is always executed whether exception is handled or not. It identifies a block of statements that needs to be executed regardless of whether or not an exception occurs within the try block.

->It is not mandatory to include a finally block at all, but if you do, it will run regardless of whether an exception was thrown and handled by the try and catch parts of the block.

->A finally block typically contains cleanup code that recovers from partial execution of a try block.

eg:

```
 try
 {
 <block that may throw exceptions>
 }
 catch (<java.lang.Exception or subclass> e)
 {
 <code to handle exception e>
 }
finally
 {
 <statements that execute with or without exception>
 }
```

Q23. Explain Immutable Objects.

Ans:

->An object is considered immutable if its state cannot change after it is constructed. Maximum reliance on immutable objects is widely accepted as a sound strategy for creating simple, reliable code.

->Immutable objects are particularly useful in concurrent applications. Since they cannot change state, they cannot be corrupted by thread interference or observed in an inconsistent state.

->The impact of object creation is often overestimated, and can be offset by some of the efficiencies associated with immutable objects. These include decreased overhead due to garbage collection, and the elimination of code needed to protect mutable objects from corruption.

->In Java, all the wrapper classes (like Integer, Boolean, Byte, Short) and String class is immutable.

Q24. Static Class in java.

Ans:

->A class can be made static only if it is a nested class.

->Nested static class doesn’t need reference of Outer class

->A static class cannot access non-static members of the Outer class

eg:

```
public class Outer {
  Java Arrays with Answers
  static class Nested_Demo {
    public void my_method() {
      System.out.println("This is my nested class");
    }
  }
  public static void main(String args[]) {
    Outer.Nested_Demo nested = new Outer.Nested_Demo();
    nested.my_method();
  }
}
```

Q25. List the commonly found tools available as part of JDK which are not part of JRE?

Ans:

i. appletviewer ->Run and debug applets without a web browser.

ii. jar->Create and manage Java Archive (JAR) files.

iii. javadoc->API documentation generator.

iv. javac -> The compiler for the Java programming language.

v. Jdb ->The Java Debugger.

Q26. Explain Method Overloading in Java.

Ans.

- Java allows you to define several methods in a class with the same name, as long as each method has a unique set of parameters. Defining two or more methods with the same name in a class is called method overloading.
- Method overloading is achieved by either:
    - Changing the number of arguments.

      Example:

      ```
      import java.util.*;

      class Adder {

          public static int add(int a,int b){
              return a+b;
          }
          public static int add(int a,int b,int c){
              return a+b+c;
          }

          public static void main(String[] args) {
              System.out.println(Adder.add(2,1));      //3
              System.out.println(Adder.add(5,2,3));    //10
          }

      }
      ```

      Here, the add() method is overloaded. These methods have the same name but accept different arguments.

    - or changing the datatype of arguments.

      Example:

      ```
      import java.util.*;

      class Adder {

          public static int add(int a,int b){
              return a+b;
          }
      ```

```
        public static double add(double a,double b){
                return a+b;
        }

        public static void main(String[] args) {
                System.out.println(Adder.add(2,1));       //3
                System.out.println(Adder.add(5.14,2.1));          //7.24
        }

}
```

Here, the add() method is overloaded. These methods have the same name but accept different datatypes of arguments.

- Method overloading is not possible just by changing the return type of methods.
- Overloaded methods may or may not have different return types, but they must differ in parameters they accept.
- Method overloading increases the readability of the program.

Q27. Compare String and StringBuilder class with methods.

Ans:

String:

1. If our context is fixed and not changing frequently, then we should go for string.
2. A string is immutable in Java.
3. We can use equals() method for comparing two strings in Java since the String Class overrides equals() method of the object class.
4. We can create a String object without using new operator.
5. String is slow as compared to StringBuilder while performing concatenations. This is because String is immutable in Java and concatenation of a two String objects involves creation of a new object
6. Since String is immutable, its length is fixed.

StringBuilder:

1. If our context is not fixed, changed frequently and thread safety is not required then we should go for StringBuilder.
2. A stringBuilder is mutable in Java.
3. StringBuilder doesn't overrides equals() method of the object class and hence equals() method cannot be used to compare two StringBuilder objects.
4. We cannot create a StringBuilder object without using new operator.

5. StringBuilder is very fast and consumes less memory than a String while performing concatenations. This is because StringBuilder is mutable.
6. But StringBuilder has setLength() method which can be used to change the StringBuilder object to the specified length.

Q28. Explain: Class, Encapsulation, Inheritance and Polymorphism.

Ans:

Class:

1. A class defines the form of an object. It specifies both the data and the code that will operate on that data.
2. A class is created by using the keyword **class**. A **class** definition creates a new data type.
3. Java uses a class specification to construct *objects*. Objects are instances of a class.
4. Thus, a class is essentially a set of plans that specify how to build an object.
5. A simplified general form of a **class** definition is shown here:

```
class classname {
  // declare instance variables
  type var1;
  type var2;
  // ...
  type varN;

  // declare methods
  type method1(parameters) {
     // body of method
  }
  type method2(parameters) {
     // body of method
  }
  // ...
  type methodN(parameters) {
     // body of method
  }
}
```

Encapsulation:

1. Encapsulation refers to the hiding of items of data and methods within an object. This is achieved by specifying them as private in the definition of the class.
2. They are accessible only through the methods defined for the class. Therefore, the only way to alter the values they contain is to call a method that does that.
3. Being able to encapsulate members of a class in this way is important for the security and integrity of class objects.
4. We may have a class with data members that can take on only particular values.
5. By hiding the data members and forcing the use of a method to set or change the values, we can ensure that only legal values are set.

Inheritance:

1. The inclusion of members of a base class in a derived class so that they are accessible in that derived class is called class inheritance
2. Inheritance is the process by which one object can acquire the properties of another object. This is important because it supports the concept of hierarchical classification
3. Without the use of hierarchies, each object would have to explicitly define all of its characteristics. Using inheritance, an object need only define those qualities that make it unique within its class.
4. It can inherit its general attributes from its parent. Thus, it is the inheritance mechanism that makes it possible for one object to be a specific instance of a more general case.
5. An inherited member of a base class is one that is accessible within the derived class. If a base class member is not accessible in a derived class, then it is not an inherited member of the derived class, but base class members that are not inherited still form part of a derived class object.

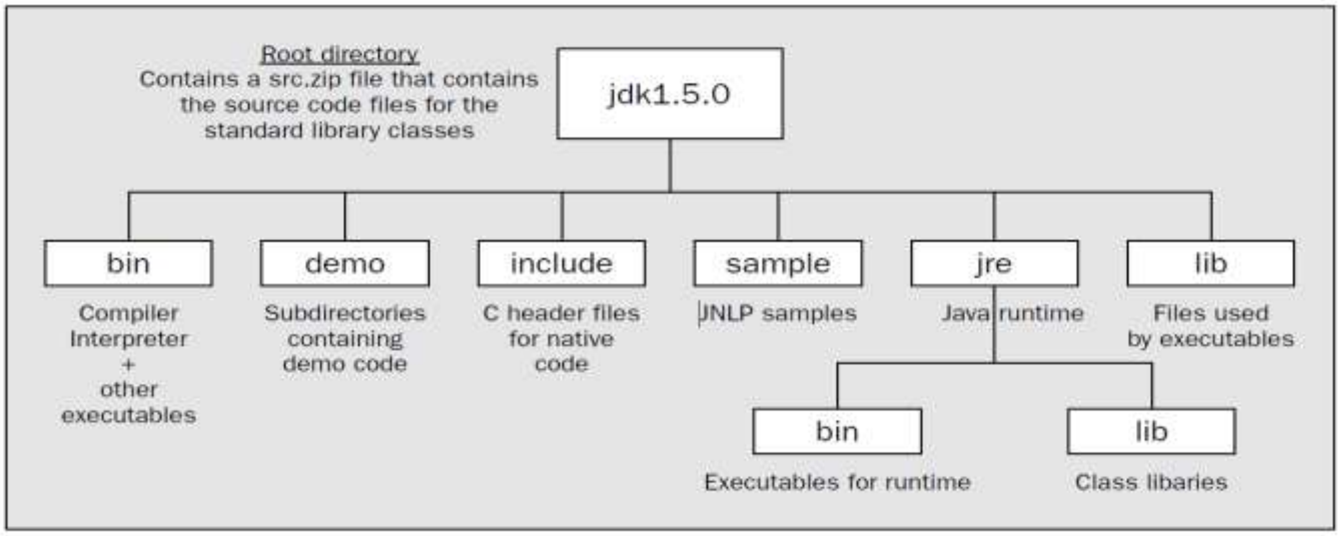
Polymorphism:

1. Polymorphism (from Greek, meaning "many forms") is the quality that allows one interface to access a general class of actions. The specific action is determined by the exact nature of the situation.
2. For example, consider a stack (which is a first-in, last-out list). You might have a program that requires three different types of stacks. One stack is used for integer values, one for floating-point values, and one for characters. In this case, the algorithm that implements each stack is the same, even though the data being stored differs. In a non-object-oriented language, you would be required to create three different sets of stack routines, with each set using different names. However, because of polymorphism, in Java you can create one general set of stack routines that works for all three specific situations.
3. Polymorphism helps reduce complexity by allowing the same interface to be used to specify a *general class of action*.
4. It is the compiler's job to select the *specific action* (i.e., method) as it applies to each situation.

Q29. Explain JDK root directory with diagram.

JDK- Java Development kit is a subset of JRE and contains everything that is in JRE, plus tools such as the compilers and debuggers necessary for developing applets and applications.

Root Directory -- Contains a src.zip file that contains the source code files for the standard library classes


Root directory
Contains a src.zip file that contains the source code files for the standard library classes
jdk1.5.0
bin
Compiler Interpreter + other executables
demo
Subdirectories containing demo code
include
C header files for native code
sample
JNLP samples
jre
Java runtime
lib
Files used by executables
bin
Executables for runtime
lib
Class libaries
Figure 1-1

Q30. Discuss five keywords of Java with example.

Java keywords are also known as **reserved words**. Keywords are particular words which acts as a key to a code. These are predefined words by Java so it cannot be used as a variable or object name. These keywords, combined with the syntax of the operators and separators, form the definition of the Java language.

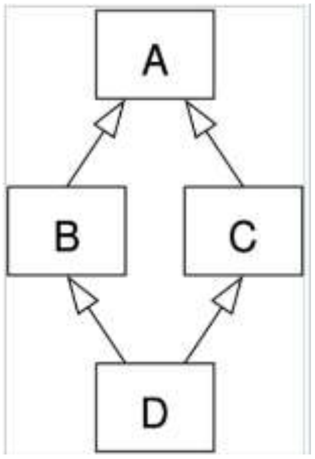
1. **abstract:** Java abstract keyword is used to declare abstract class. Abstract class can provide the implementation of interface. It can have abstract and non-abstract methods.
2. **boolean:** Java boolean keyword is used to declare a variable as a boolean type. It can hold True and False values only.
3. **break:** Java break keyword is used to break loop or switch statement. It breaks the current flow of the program at specified condition.
4. **byte:** Java byte keyword is used to declare a variable that can hold an 8-bit data values.
5. **case:** Java case keyword is used to with the switch statements to mark blocks of text.
6. **enum:** Java enum keyword is used to define a fixed set of constants. Enum constructors are always private or default.

7. **extends:** Java extends keyword is used to indicate that a class is derived from another class or interface.
8. **final:** Java final keyword is used to indicate that a variable holds a constant value. It is applied with a variable. It is used to restrict the user.
9. **finally:** Java finally keyword indicates a block of code in a try-catch structure. This block is always executed whether exception is handled or not.

Q31 Discuss the diamond problem with reference to inheritance.

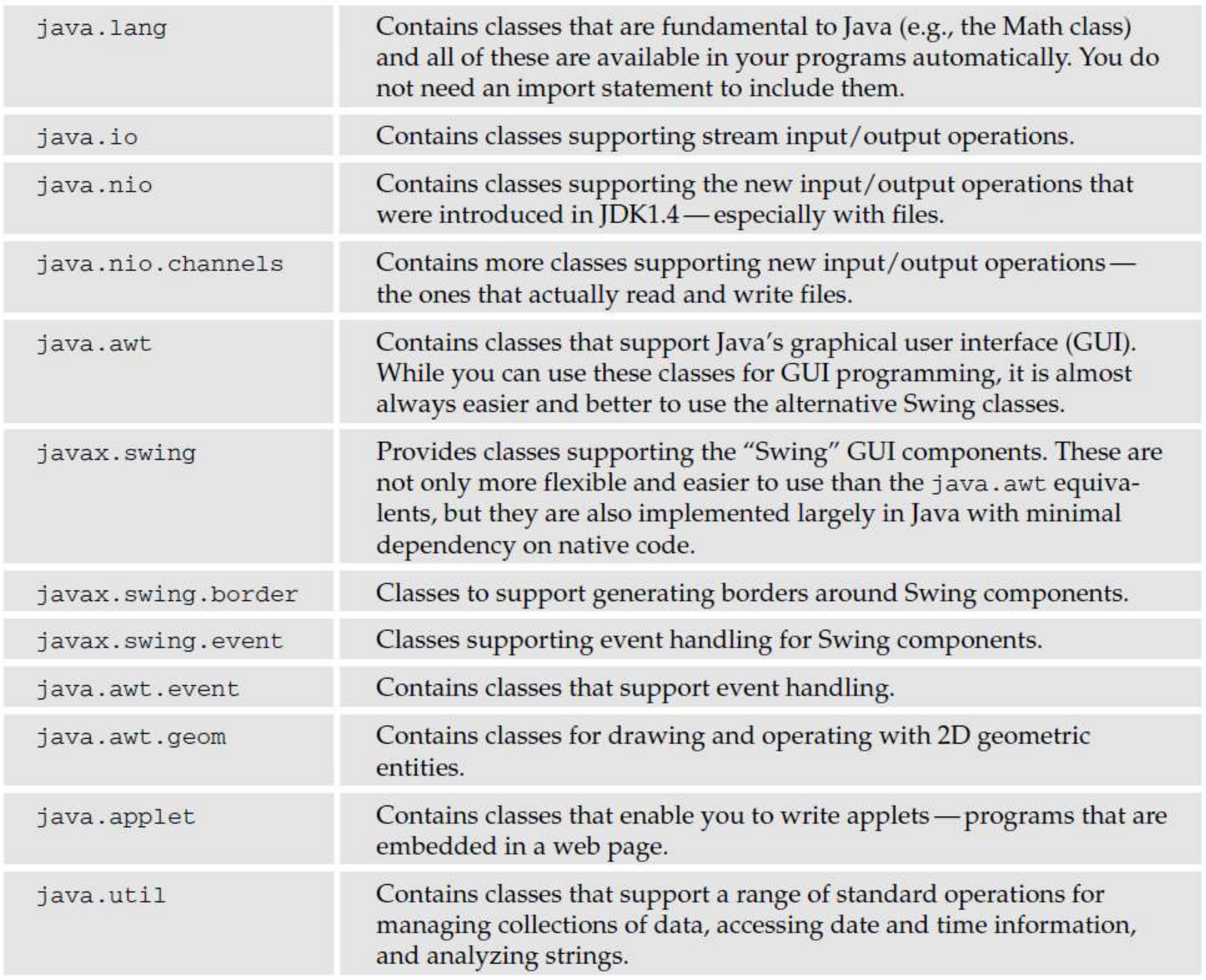
The "**diamond problem**" is an ambiguity that arises when two classes B and C inherit from A, and class D inherits from both B and C.

It is called the "diamond problem" because of the shape of the class inheritance diagram in this situation. In this case, class A is at the top, both B and C separately beneath it, and D joins the two together at the bottom to form a diamond shape.



Q32 Discuss any 5 standards packages in java.

All of the standard classes that are provided with Java are stored in standard packages. There is a substantial and growing list of standard packages (more than 150 in JDK 5) but some of the ones you may hear about most frequently are:



`java.lang`	Contains classes that are fundamental to Java (e.g., the Math class) and all of these are available in your programs automatically. You do not need an import statement to include them.
`java.io`	Contains classes supporting stream input/output operations.
`java.nio`	Contains classes supporting the new input/output operations that were introduced in JDK1.4—especially with files.
`java.nio.channels`	Contains more classes supporting new input/output operations—the ones that actually read and write files.
`java.awt`	Contains classes that support Java's graphical user interface (GUI). While you can use these classes for GUI programming, it is almost always easier and better to use the alternative Swing classes.
`javax.swing`	Provides classes supporting the "Swing" GUI components. These are not only more flexible and easier to use than the `java.awt` equivalents, but they are also implemented largely in Java with minimal dependency on native code.
`javax.swing.border`	Classes to support generating borders around Swing components.
`javax.swing.event`	Classes supporting event handling for Swing components.
`java.awt.event`	Contains classes that support event handling.
`java.awt.geom`	Contains classes for drawing and operating with 2D geometric entities.
`java.applet`	Contains classes that enable you to write applets—programs that are embedded in a web page.
`java.util`	Contains classes that support a range of standard operations for managing collections of data, accessing date and time information, and analyzing strings.

Q33 Explain finalize().

Ans.

The **finalize() method** is defined in **Object class** which is the super most class in Java. This method is called by **Garbage collector** just before they destroy the object from memory. The **Garbage collector** is used to destroy the object which is **eligible for garbage collection**.

finalize() method is a protected and non-static method of java.lang.Object class. This method will be available in all objects you create in java.

This method is used to perform some final operations or clean up operations on an object before it is removed from the memory. you can override the finalize() method to keep those operations you want to perform before an object is destroyed. Here is the general form of finalize() method.

Example.

```
protected void finalize() {
   //Keep some resource closing operations here
}
```

Another use for the finalize() method is to record the fact that the object has been destroyed.

Q34 How the finalize() method works with Garbage collection?

Ans.

The **finalize() method** is called by the **garbage collector** for an object when **garbage collection** determines that there are no more references to the object. The **garbage collection** determines each object of the **class**, if there is no reference for the object it means it should be destroyed.

Let's understand with an example:

```
public class Example

{

public static void main(String[] args)

{

String string1 = "Hello";

string1 = null;

}

}
```

1.When **String** object ***string1*** hold the value "Hello". It means ***string1*** is a reference of the **String** object.
2. When **String** object ***string1*** holds the **NULL.** It means does not have any reference then it is eligible for **garbage collection**.
3. The **Garbage collector** makes a call to **finalize() method** to perform some activity before destroying the object.

Q35 Boxing , Autoboxing and Unboxing

Ans.

Boxing- boxing is converting primitive data type to reference data type or wrapping primitive types into a reference type explicitly

auto boxing is boxing done implicity i.e by JVM

Example.

Integer a= Integer(5);

AutoBoxing- Automatic conversion of primitive types to the object of their corresponding wrapper classes is known as autoboxing. For example – conversion of int to Integer, long to Long, double to Double etc.

Example.

->Integer a =5;

```
class BoxingExample1{

  public static void main(String args[]){

    int a=50;

        Integer a2=new Integer(a);//Boxing

        Integer a3=5;//Boxing

        System.out.println(a2+" "+a3);

 }

}
```

O/P-> 50 5

Unboxing - Automatically converting an object of a wrapper class to its corresponding primitive type is known as unboxing. For example – conversion of Integer to int, Long to long, Double to double etc.

Example.

```
class UnboxingExample1{

  public static void main(String args[]){

    Integer i=new Integer(50);

        int a=i;

        System.out.println(a);

 }

}   O/P-> 50
```

Q36 Compare boxing and unboxing in Java.

	Boxing	**Unboxing**
Meaning	boxing is converting primitive data type to reference data type or wrapping primitive types into a reference type explicitly	Automatically converting an object of a wrapper class to its corresponding primitive type is known as unboxing.
Type Of Conversion	Implicit Conversion	Explicit Conversion

Example	Integer a= Integer(5);	conversion of Integer to int, Long to long, Double to double etc.
		
		class UnboxingExample1{
		  public static void main(String args[]){
		    Integer i=new Integer(50);
		        int a=i;
		
		        System.out.println(a);
		 }
		}
		
		o/p-> 50

Q37 Define final keyword or what do u understand by final keyword. Discuss the purpose of final keyword and its advantages.

Ans.

DEFINE -- Final is a keyword or reserved word in Java and can be applied to member variables, methods, class and local variables in Java. Once you make a reference final you are not allowed to change that reference and compiler will verify this and raise a **compilation error** if you try to re-initialized **final variables in Java**.

Example:

final int FEET_PER_YARD = 3; // Constant values

final double MM_PER_INCH = 25.4; // that cannot be changed

Java final keyword can be used in many context. Final can be:

1.variable

2.method

3.class

1.variable

The keyword final tells the compiler that you do not want the value of this variable to be changed, so the compiler will check that this variable is not modified anywhere in your program.

If you make any variable as final, you cannot change the value of final variable(It will be constant).

```
// a final variable
final int THRESHOLD = 5;
```

2.method

When a method is declared with *final* keyword, it is called a final method. A final method cannot be overridden. The Object class does this—a number of its methods are final.We must declare methods with final keyword for which we required to follow the same implementation throughout all the derived classes.

Examples

```
 class A
{
    final void m1()
    {
        System.out.println("This is a final method.");
    }
}

class B extends A
{
    void m1()
    {
        // COMPILE-ERROR! Can't override.
        System.out.println("Illegal!");
    }
}
```

3.class

When a class is declared with *final* keyword, it is called a final class. A final class cannot be extended(inherited). There are two uses of a final class :

1. One is definitely to prevent inheritance, as final classes cannot be extended. For example, all Wrapper Classes like Integer,Float etc. are final classes. We can not extend them.

```
final class A
{
     // methods and fields
}
// The following class is illegal.
class B extends A
{
    // COMPILE-ERROR! Can't subclass A
}
```

2. The other use of final with classes is to create an immutable class like the predefined String class. You cannot make a class immutable without making it final.