# Java Descriptive Questions & Answers

1. **What do you mean by object and class?**

**Object**s are all around us. Anything can be thought of as an Object. An object is an instance of a class. An Object **is** a module that has both state and behaviour.

A **Class** **is** nothing but a blue print for creating different objects which **defines** the shape and nature of an object. Class is a term that **describes** a specification for a collection of objects with common properties. A Class **is** a Template for an object that **defines** a new data type.

2. **What do you mean by inheritance?**

**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. A very important fact to remember is that Java only supports only single inheritance. This means that a class cannot extend more than one class.

3. **What do you mean by narrowing and widening? Give an example of them.**

Conversion from a number with a smaller range of magnitude ( like **int** to **long** or **long** to **float** ) is called **widening**. The goal of widening conversions is to produce no change in the magnitude of the number while preserving as much of the precision as possible. For example, converting the **int** 2147483647 to **float** produces 2.14748365e9 or 2,147,483,650. The difference is usually small, but it may be significant.

Conversely, conversion where there is the possibility of losing information about the magnitude of the number ( like **long** to **int** or **double** to **long** ) is called **narrowing**. With narrowing conversions, some information may be lost, but the nearest representation is found whenever possible. For example, converting the **float** 3.0e19 to **long** yields -9223372036854775807, a very different number.

4. **What do you mean by 'super( )' and 'this( )' keyword?**

**this( )** & **super( )** is a keyword.
"**this( )**" is used to invoke a constructor of *a same class*, its pointing the same class object.
"**super( )**" is used to invoke a super class constructor and accessing the super class constructor.

5. **Write down the difference between overloading and overridden.**

**Overriding** - *same method names* with *same arguments* and *same return types* associated in a class and its **subclass.**

**Overloading** - *same method name* with *different arguments may or may not be same return type* written in the **same class itself**.

6. **Where we use abstract and final keyword?**

**Abstract:** **abstract keyword** can be applied to class, interface and method.
class_ Contains unimplemented methods and cannot be instantiated.
interface_ All interfaces are abstract. Optional in declarations.
method_ No body, only signature. The enclosing class is abstract

**Final:** **final keyword** can be applied to variables, method, class and field.
class_ Can not be sub classed.
method_ Can not be overridden and dynamically looked up.
field_ Can not change its value. static final fields are compile-time constants.
Variable_ Can not change its value.

## 7. Write down the difference between interface and abstract class?

**Abstract:**

- When one or more methods of a class are abstract.
- When the class is a subclass of an abstract class and does not provide not provide any implementation details or method body to any of the abstract methods.
- When a class implements an interface and does not provide implement implementation details or method body to any of the abstract methods.

**interface:**

- An interface is not a class but a collection of abstract methods. Writing an interface is similar to writing a class, but they are two different concepts. A class **describes** the attributes and behaviors of an object. An interface **contains** behaviors that a class implements. A Class uses the **implements** keyword to implement an interface.
- We cannot instantiate an interface. An interface does not contain any constructors and instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

## 8. What is the advantage of package? Write down the default package.

**Packages:**

Package is a folder to organize our classes and interfaces. Package can also contain other packages representing a smaller, more specific grouping of classes.

- Packages are useful to :
  - ✓ Organize classes into smaller units so that it is easy to locate.
  - ✓ Helps to avoid naming conflicts.
  - ✓ Protect our classes, data and methods in a larger way on a class to class basis.
  - ✓ Identify our classes with the help of the package names.

The classes of **default package** support the basic language features and the handling of arrays and strings. Classes in this package are always available directly in our programs by default because this package is always automatically loaded with our program. The JDK includes the default package name as java.lang.

## 9. What do you mean by logical and short circuit operator?

The **logical operator** is & and the **short circuit operator** is &&.

When & is used it will evaluate both the expressions regardless of the fact that it finds first expression as FALSE and only then will it give an answer. Whereas if && was used in place of &, after it had evaluated first expression and had found result of first expression as FALSE, it would not have evaluated second expression.

## 10. Write down the difference between wrapper class and data type?

Java has eight primitive **data types**: four integer types for whole-valued signed numbers (short, byte, int, long), two floating point types for representing numbers with fractional precision (float, double), one character type for Unicode encoding (char), and one boolean type for representing logical values (boolean).

Java provides specialized classes corresponding to each of the primitive data types. These are called **wrapper classes**. They are e.g. Integer, Character and Double etc. It is sometimes easier to deal with primitives as objects. Moreover most of the collection classes store objects and not primitive data types. And also the wrapper classes provide many utility methods also. Because of these reasons we need wrapper classes. And since we create instances of these classes we can store them in any of the collection classes and pass them around as a collection. Also we can pass them around as method parameters where a method expects an object.

## 11. Write down the difference between instance variable and static variable?

**Instance variables** are declared in a class, but outside a method. They are also called member or field variables. When an object is allocated in the heap, there is a slot in it for each instance variable value. Therefore an instance variable is created when an object is created and destroyed when the object is destroyed. Visible in all methods and constructors of the defining class, should generally be declared private, but may be given greater visibility.

**Class/static variables** are declared with the static keyword in a class, but outside a method. There is only one copy per class, regardless of how many objects are created from it. They are stored in static memory. It is rare to use static variables other than declared final and used as either public or private constants.

## 12. How many ways we can declare an array and what are they?

A Java **array** is an ordered collection of primitives, object references, or other arrays. Java arrays are homogeneous: except as allowed by polymorphism, all elements of an array must be of the same type. Each variable is referenced by array name and its index. Arrays may have one or more dimensions.

We can **declare arrays** in the following ways…

```
    typevar[] name = new type[size];
    typevar name [] = new type[size];
    typevar name [] = {data1, data2, data3};
    typevar name [] = new type[] {data1, data2, data3};
```

## 13. What do you mean by default constructor? How can we express default constructor?

If we don't define any constructor for our class, the compiler will supply a **default constructor** in the class. The default constructor is also described as the no-args constructor, because it requires no arguments to be specified when it is called.

```
class Student{
    long Roll;
    String Name;

    public Student(){
        Roll=101;
        Name="Test";
        System.out.println("Roll is :"+Roll);
        System.out.println("Name is :"+Name);
    }
}
public class ConstractorEmpty{
    public static void main (String args[]){
        Student s=new Student();
    }
}
```

## 14. What do you mean by statement and keyword?

**Statements** are similar to sentences in the English language. A sentence forms a complete idea which can include one or more clauses. Likewise, a statement in Java forms a complete command to be executed and can include one or more expressions. There are three main groups that encompass the different kinds of statements in Java:

- Expression statements: these change values of variables, call methods and create objects.
- Declaration statements: these declare variables.
- Control flow statements: these determine the order that statements are executed.

Examples:

```
 int number; //declaration statement
 number = 4; //expression statement
 if (number < 10) //control flow statement
 {
System.out.println(number + " is less than ten");//expression statement
 }
```

In the Java programming language, a **keyword** is one of the reserved words which have a predefined meaning in the language; because of this, programmers cannot use keywords as names for variables, methods, classes, or as any other identifier. There are more than 50 reserved word in Java programming language.

### 15. Write down seven keywords in java?

The following is a **list of the keywords** in Java:

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

Reserved words *for literal values*

| | | |
|---|---|---|
| False | null | true |

### 16. Do you think java support multiple-inheritance? If no what we use for its substitution an why?

In Java, classes do not support **multiple inheritance**, only interfaces do.
When we use this capability, if two or more super-interfaces declare a method with the same signature- that is, with identical names and parameters- the method must have the same return type in all the interfaces that declare it. If they don't, the compiler will report an error. This is because it would be impossible for a class to implement both methods, as they have the same signature. A single definition of class will satisfy all the interfaces, that is, *every method in a class must have unique signature, and the return type not part of it*.

### 17. Which class is the mother of all class? Write down the difference between wrapper class and data type?

Object class is called **the mother class** of all class.

Java has eight primitive **data types**: four integer types for whole-valued signed numbers (short, byte, int, long), two floating point types for representing numbers with fractional precision (float, double), one character type for Unicode encoding (char), and one boolean type for representing logical values (boolean).

Java provides specialized classes corresponding to each of the primitive data types. These are called **wrapper classes**. They are e.g. Integer, Character and Double etc. It is sometimes easier to deal with primitives as objects. Moreover most of the collection classes store objects and not primitive data types. And also the wrapper classes provide many utility methods also. Because of these reasons we need wrapper classes. And since we create instances of these classes we can store them in any of the collection classes and pass them around as a collection. Also we can pass them around as method parameters where a method expects an object.

## 18. Write down the difference between String class and String Buffer class?

**String Class:**

- A series of characters and is different from an array of characters.
- It is an object.
- Strings are constant once created cannot be changed.

**StringBuffer Class:**

- StringBuffer class provides various methods to manipulate string object.
- Objects of StringBuffer class are growable and flexible.
- Characters or strings can be inserted in between in the StringBuffer object as well as they can be appended at the end.
- The capacity of string buffer can be changed with ensureCapacity() method provided in the class.

## 19. What do you mean by indexOf( ) and charAt( );

**charAt( )** public char charAt(int index)
Returns the character at the specified index. An index ranges from 0 to length( ) - 1. The first character of the sequence is at index 0, the next at index 1, and so on, as for array indexing.
**Specified by:** charAt in interface CharSequence, **Parameters:** index - the index of the character.

**indexOf( )** public int indexOf(int ch)
Returns the index within this string of the first occurrence of the specified character. If a character with value ch occurs in the character sequence represented by this String object, then the index of the first such occurrence is returned -- that is, the smallest value k such that: this.charAt(k) == ch
is true. If no such character occurs in this string, then -1 is returned.
**Parameters:** ch - a character.

**indexOf()** public int indexOf(int ch, int fromIndex)
Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index. If a character with value ch occurs in the character sequence represented by this String object at an index no smaller than fromIndex, then the index of the first such occurrence is returned--that is, the smallest value k such that:
(this.charAt(k) == ch) && (k >= fromIndex)
is true. If no such character occurs in this string at or after position fromIndex, then -1 is returned. There is no restriction on the value of fromIndex. If it is negative, it has the same effect as if it were zero: this entire string may be searched. If it is greater than the length of this string, it has the same effect as if it were equal to the length of this string: -1 is returned.
**Parameters:** ch - a character. fromIndex - the index to start the search from.

## 20. Write down the difference between equels() and "= =”?

The **equals()** method and **= = operator** perform two different operations. The equals() method **compares** *the character inside a String object*. The **= =** operator **compares** *two object reference* to see whether they refer to the same instance.

The following program shows the differences:

```
public class Main {

  public static void main(String args[]) {
    String s1 = "java2s.com";
    String s2 = new String(s1);

    System.out.println(s1 + " equals " + s2 + " -> " + s1.equals(s2));
    System.out.println(s1 + " == " + s2 + " -> " + (s1 == s2));
  }
}
```

Here is the output of the preceding example:

```
java2s.com equals java2s.com -> true
java2s.com == java2s.com -> false
```

## 21. What do you mean by Exception Handling? What is the purpose of Exception Handling?

The ideal time to catch an error is at compile time before trying to run program. However, not all error can be detected at compile time; rest of the problems must be handled at run time. Exception is a special type of error. It arises at run time in a code sequence. It is because of abnormal conditions that occur at the time of executing the program. When an exception is occurred, an object, which represents that exception, is created and passed to the method, where exception has occurred. Runtime system is then responsible to find some code to handle error and clean up. In general, reading, writing and debugging code becomes much clearer with exception than when using the way of error handling.

An exception is a problem that arises during the execution of a program. An exception can occur for many different reasons, including the following:

- A user has entered invalid data.
- A file that needs to be opened cannot be found.
- A network connection has been lost in the middle of communications, or the JVM has run out of memory.

Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

To understand how exception handling works in Java, you need to understand the three categories of exceptions:

- **Checked exceptions:** A checked exception is an exception that is typically a user error or a problem that cannot be foreseen by the programmer. For **Example**, if a file is to be opened, but the file cannot be found, an exception occurs. These exceptions cannot simply be ignored at the time of compilation.
- **Runtime exceptions:** A runtime exception is an exception that occurs that probably could have been avoided by the programmer. As opposed to checked exceptions, runtime exceptions are ignored at the time of compilation.
- **Errors:** These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For **Example**, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

Exception can be handled with the five keywords provided by the Java language. They are as follows:

- ➢ **Try** (If a matching catch block is found, then exception is handled in that block otherwise Java Runtime Environment handles the exception.)
- ➢ **Catch** (Catch block as the name suggests, catches the exceptions thrown in the try block.)
- ➢ **Throw** (Throw keyword is used to indicate that exception has occurred)
- ➢ **throws** (Throws clause or keyword is used to indicate that main method may throw IOException or NumberFormatException.)
- ➢ **finally** (Finally block is the block in which we find statements which return resources to the system and other statements for printing any message.)

## 22. What do you mean by Assertion? Declare an assertion statement?

Assert is used during program development to create an assertion which is a condition that should be true during the execution of program. **Assertions** are often used during testing to verify that some expected condition is actually met.
A simple assertion is a statement of the form

```
assert logical_expression;
```

**Example**:

```
If(daysInMonth == 31){
        System.out.println("Jan, Mar, May, Jul, Aug, Oct, or Dec");
} else if(daysInMonth == 30){
        System.out.println("Apr, Jun, Sep, or Nov");
} else {
        assert daysInMonth == 28 || daysInMonth == 29;
        System.out.println("Feb");
}
```

## 23. Write down the difference between Assertion and Exception Handling?

Assert is used during program development to create an assertion which is a condition that should be true during the execution of program. **Assertions** are often used during testing to verify that some expected condition is actually met. A simple assertion is a statement of the form

```
assert logical_expression;
```

**Exception** is a special type of error. It arises at run time in a code sequence. It is because of abnormal conditions that occur at the time of executing the program. When an exception is occurred, an object, which represents that exception, is created and passed to the method, where exception has occurred. Runtime system is then responsible to find some code to handle error and clean up.

## 24. Write down the difference between throw and throws?

**Throw:**

- Throw keyword is used to indicate that exception has occurred.
- The operand of throw is an object of a class, which is derived from class Throwable

**Throws:**

- Throws clause or keyword is used to indicate that main method may throw IOException or NumberFormatException.

## 25. What do you mean by enum?

An ***enum type*** is **a type** whose *fields* consist of a fixed set of constants. Common examples include compass directions (values of NORTH, SOUTH, EAST, and WEST) and the days of the week. Because they are constants, the names of an enum type's fields are in uppercase letters. In the Java programming language, you define an enum type by using the enum keyword. For example, we would specify a days-of-the-week enum type as:
public enum Day {SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY}

## 26. What do you mean by static variable and instance variable and local variable?

**Class/static variables** are declared with the static keyword in a class, but outside a method. There is only one copy per class, regardless of how many objects are created from it. They are stored in static memory. It is rare to use static variables other than declared final and used as either public or private constants.
**Instance variables** are declared in a class, but outside a method. They are also called member or field variables. When an object is allocated in the heap, there is a slot in it for each instance variable value. Therefore an instance variable is created when an object is created and destroyed when the object is destroyed. Visible in all methods and constructors of the defining class, should generally be declared private, but may be given greater visibility.
**Local variables** are declared in a method, constructor, or block. When a method is entered, an area is pushed onto the call stack. This area contains slots for each local variable and parameter. When the method is called, the parameter slots are initialized to the parameter values. When the method exits, this area is popped off the stack and the memory becomes available for the next called method. Parameters are essentially local variables which are initialized from the actual parameters. Local variables are not visible outside the method.

## 27. Write some method of string buffer class?

Some of the basic **StringBuffer methods** like;

- **capacity** (): Returns the current capacity of the String buffer.
- **length** (): Returns the length (character count) of this string buffer.
- **charAt** (int index): The specified character of the sequence currently represented by the string buffer, as indicated by the index argument, is returned.
- **setCharAt** (int index, char ch): The character at the specified index of this string buffer is set to ch
- **toString** ():Converts to a string representing the data in this string buffer
- **insert** (int offset, char c): Inserts the string representation of the char argument into this string buffer.
  *Note* that the StringBuffer class has got many overloaded 'insert' methods which can be used based on the application need.
- **delete** (int start, int end): Removes the characters in a substring of this StringBuffer

- **replace** (int start, int end, String str): Replaces the characters in a substring of this StringBuffer with characters in the specified String.
- **reverse** (): The character sequence contained in this string buffer is replaced by the reverse of the sequence.
- **append** (String str): Appends the string to this string buffer.
  *Note* that the StringBuffer class has got many overloaded 'append' methods which can be used based on the application need.
- **setLength**(int newLength): Sets the length of this String buffer.

## 28. What is OOP? Feature of OOP?

**OPP- Object Oriented Programming**
Object Oriented Programming is a method of implementation in which programs are organized as cooperative collection of objects, each of which represents an instance of a class, and whose classes are all members of a hierarchy of classes united via inheritance relationships. Object Oriented programs are easier to understand and less time consuming to maintain and extend.

**Four principles of Object Oriented Programming are:**
**Abstraction:** Abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer.
**Encapsulation:** Encapsulation is the process of compartmentalizing the elements of an abstraction that constitute its structure and behavior; encapsulation serves to separate the contractual interface of an abstraction and its implementation.

- Hides the implementation details of a class.
- Forces the user to use an interface to access data
- Makes the code more maintainable.

**Inheritance:** This is the process by which one object acquires the properties of another object.
**Polymorphism:** This is the existence of the classes or methods in different forms or single name denoting different implementations.

## 29. Write down the difference between while and do while?

Loops execute a block of code a specified number of times, or while a specified condition is true.
**while:** The while loop, loops through a block of code while a specified condition is true.
**do...while:** The do...while loop is a variant of the while loop. This loop will execute the block of code at least ONCE, and then it will repeat the loop as long as the specified condition is true.

## 30. What do you mean by type casting?

**Type Casting** is used to convert the value of one type to another. There are two types of typecasting.
**Explicit casting:** Explicit casting is the process in which the compliers are specifically informed to about transforming the object.

```
Example:
long i = 700.20;
```

**Implicit casting:** Implicit casting is the process of simply assigning one entity to another without any transformation guidance to the compiler. This type of casting is not permitted in all kinds of transformations and may not work for all scenarios.

```
Example:
int i = 1000;
long j = i; //Implicit casting
```

## 31. What do you mean by Auto-Boxing?

**Autoboxing**, introduced in Java 5, is the automatic conversion the Java compiler makes between the primitive (basic) types and their corresponding object wrapper classes (eg, int and Integer, double and Double, etc). The underlying code that is generated is the same, but autoboxing provides a sugar coating that avoids the tedious and hard-to-read casting typically required by Java Collections, which can not be used with primitive types.

## 32. What do you mean by tokenizing?

The split( ) method in the String class is specifically for splitting into tokens. This is done in a single step, returning all the tokens from a string as an array of String objects. This procedure is known as **Tokenizing**.

```
Example:
String text = "to be or not to be, that is the question."; // any string.
String[] words = text.split("[, .]",0); // split("[Delimiters]",Limit)
```

## 33. Write down the difference between finalize () and finally?

**finally:** finally is the last clause in a try...catch block. It is a block of statements that is executed irrespective if or if not an exception was caught in the preceding try block. Each try contain only one finally blocks not more than one.
**finalize():** This is method used to release the occupied memory /any expensive resources including native peer objects, file/device/database connections. finally method must be protected or public otherwise compile time error.

## 34. Describe Standard Package?

All of the standard classes that are provided with java are stored in a **standard package**. There is a substantial and growing list of standard packages. Some of those are as follows:

**java.lang** _ contains classes that are fundamental to Java.
**java.io** _ contains classes supporting stream input/output operations.
**java.nio** _ contains classes supporting stream input/output operations.
**java.nio.channels** _ contains classes supporting stream input/output operations.
**java.awt** _ contains classes that support Java's GUI.
**javax.swing** _ contains classes that support Java's GUI.
**java.awt.evnt** _ 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 us to write applets.
**java.util** _ contains classes that support a range of standard operations for managing collections of data, accessing date and time information and analyzing strings.

## 35. What do you mean by Recursion?

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

## 36. What do you mean by parameter and argument?

A ***parameter*** **defines** the type of value that **can be passed** to the method when it is called. A parameter has a name and a type that **appears** in the parameter list in the definition of a method.
An ***argument*** is a value that is passed to a method when it is executed. The value of the arguments is referenced by the parameter name during execution of the method.

## 37. What do you mean by continue and break?

A **continue;** statement is used to end the current loop iteration and return control to the loop statement. A continue statement skips to the end of a loop's body and evaluates the boolean-expression that controls the loop. In particular, in a for loop, the increment expression is executed before examining the boolean-expression.

- A continue only has meaning inside: for, while, do-while.
- If the continue keyword is followed by a label of an enclosing loop, execution skips to the end of the labeled loop.
- If no label is specified, then execution skips to the end of the loop in which the continue is located within.

A **break;** statement results in the termination of the statement to which it applies (switch, for, do, or while). A labeled **break** statement can be used to exit from any block, whereas an unlabeled **break** can only be used inside a loop control structure or switch structure.

- An unlabeled break terminates the innermost for, while, do or switch.
- To terminate an outer loop, use a labeled break, where the outer loop statement is proceeded by the label name.

### 38. What is JVM? Describe.

**JVM** stand for Java Virtual Machine. The JVM is the environment in which Java programs execute. It is a software that is implemented on top of real hardware and operating system.
When the source code (.java files) is compiled, it is translated into byte codes and then placed into (.class) files. The JVM executes these byte codes. So Java byte codes can be thought of as the machine language of the JVM. The JVM is an interpreter for bytecode. The JVM is a software that can be ported onto various hardware-based platforms.

### 39. Describe "Java is a Platform Independent Language".

Java is compiled to an intermediate form called byte-code. A Java program never really executes natively on the host machine. Rather a special native program called the Java interpreter reads the byte code and executes the corresponding native machine instructions. Java is platform independent as JVM compiles source code to its byte code which is then interpreted to object code. Thus any machine having a java compiler can execute that byte code. This does not depend on the hardware or the OS of the system.

### 40. Describe Naming Rule for a variable?

**Rules for naming variables**:

- All variable names must begin with a letter of the alphabet, an underscore, or ( _ ), or a dollar sign ($). The convention is to always use a letter of the alphabet. The dollar sign and the underscore are discouraged.
- After the first initial letter, variable names may also contain letters and the digits 0 to 9. No spaces or special characters are allowed.
- The name can be of any length.
- Uppercase characters are distinct from lowercase characters. Using ALL uppercase letters are primarily used to identify constant variables. Remember that variable names are case-sensitive.
- You cannot use a java keyword (reserved word) for a variable name.

## Some more…

**** What is difference between int and integer?**
int is a primitive datatype like char float double boolean short long. Integer is a wrapper class. Every primitive data type is having its own wrapper class like Float Double Boolean Long Character Short
int is in primitive data type and integer is an object or an wrapper class type of data and ex used in vectors.
Integer is similar to object .when u have to send or get an int as an object u use Integer eg:
int i 10;display{new Integer(i)}

****What do you mean by 'isA' and 'hasA' relationship?**
The 'isA' relationship is expressed with inheritance and 'hasA' relationship is expressed with composition.

IS-A Relationship:
IS-A is a way of saying : This object is a type of that object. The extends keyword is used to achieve inheritance. With use of extends keyword the subclasses will be able to inherit all the properties of the superclass except for the private properties of the superclass. The implements keyword is used by classes to inherit from interfaces. Interfaces can never be extended. **is a --- House is a Building**

```
public class Animal{
}
public class Mammal extends Animal{
}
public class Reptile extends Animal{
}
public class Dog extends Mammal{
}
```

HAS-A relationship:
These relationships are mainly based on the usage. This determines whether a certain class HAS-A certain thing. This relationship helps to reduce duplication of code as well as bugs. **has a -- House has a bathroom**

```
public class Vehicle{}
public class Speed{}
public class Van extends Vehicle{
        private Speed sp;
}
```

This shows that class Van HAS-A Speed. By having a separate class for Speed we do not have to put the entire code that belongs to speed inside the Van class., which makes it possible to reuse the Speed class in multiple applications.

****What do you understand by Final Class ?**

Java allows us to declare our class as final; that is, the class declared as final cannot be subclassed. There are two reasons why one wants to do this: security and design purpose.

*Security:* To subvert systems, One of the mechanism hackers do use is, create subclasses of a class and substitute their class for the original. The subclass looks same as the original class but it will perform vastly different things, causing damage or getting into private information possibly. To prevent this subversion, you should declare your class as final and prevent any of the subclasses from being created.

*Design:* Another reason us to declare a class as final is object-oriented design reason. When we feel that our class is "perfect" and should not have subclasses further. We can declare our class as final.

**** What are methods and how are they defined? Describe main method.**

Methods are functions that operate on instances of classes in which they are defined. Objects can communicate with each other using methods and can call methods in other classes. Method definition has four parts. They are name of the method, type of object or primitive type the method returns, a list of parameters and the body of the method. A method's signature is a combination of the first three parts mentioned above.

**main( )** method is called the brain of the Java application. It is necessary to specify the name of the class which is required to run while running a Java application using the Java interpreter. Interpreter will do invokes the main( ) method defined in the class. The main( ) method will control the program flow, allocates all the resources which are needed, and will run any of the methods that do provide the functionality to the application.

**** What is the advantage of package? Write down the default package.**

A Package is a collection of classes and interfaces that provides a high-level layer of access protection and name space management. In short it is a folder to organize your classes and interfaces
Packages are useful to:

- organize classes into smaller units so that it is easy to locate
- helps to avoid naming conflicts
- protect your classes, data and methods in a larger way on a class to class basis identify classes with the help of the package names Can also contain other packages representing a smaller, more specific grouping of classes to make a class member of the package - begin your code with a package declaration

Default package is java.lang

****What is the difference between exception and error?**

The exception class defines mild error conditions that your program encounters. Exceptions can occur when trying to open the file, which does not exist, the network connection is disrupted, operands being manipulated are out of prescribed ranges, the class file you are interested in loading is missing. The error class defines serious error conditions that you should not attempt to recover from. In most cases it is advisable to let the program terminate when such an error is encountered.

****What is Garbage Collection and how to call it explicitly?**

When an object is no longer referred to by any variable, java automatically reclaims memory used by that object. This is known as garbage collection. System. gc() method may be used to call it explicitly.

****What are Transient and Volatile Modifiers?**

*Transient:* The transient modifier applies to variables only and it is not stored as part of its object's Persistent state. Transient variables are not serialized.

*Volatile:* Volatile modifier applies to variables only and it tells the compiler that the variable modified by volatile can be changed unexpectedly by other parts of the program

****What is the difference between superclass and subclass?**
A super class is a class that is inherited whereas sub class is a class that does the inheriting.

****What modifiers may be used with top-level class?**
public, abstract and final can be used for top-level class.

****What is interface and its use?**

Interface is similar to a class which may contain method's signature only but not bodies and it is a formal set of method and constant declarations that must be defined by the class that implements it.
Interfaces are useful for:

- Declaring methods that one or more classes are expected to implement
- Capturing similarities between unrelated classes without forcing a class relationship.
- Determining an object's programming interface without revealing the actual body of the class.

****What is the difference between Integer and int?**

a. Integer is a class defined in the java.lang package, whereas int is a primitive data type defined in the Java language itself. Java does not automatically convert from one to the other.
b. Integer can be used as an argument for a method that requires an object, whereas int can be used for calculations.

****What are wrapper classes? Why do we need wrapper classes?**

Java provides specialized classes corresponding to each of the primitive data types. These are called wrapper classes. They are e.g. Integer, Character, Double etc.
It is sometimes easier to deal with primitives as objects. Moreover most of the collection classes store objects and not primitive data types. And also the wrapper classes provide many utility methods also. Because of these reasons we need wrapper classes. And since we create instances of these classes we can store them in any of the collection classes and pass them around as a collection. Also we can pass them around as method parameters where a method expects an object.

****What are runtime exceptions?**
Runtime exceptions are those exceptions that are thrown at runtime because of either wrong input data or because of wrong business logic etc. These are not checked by the compiler at compile time.

****What is an abstract class?**
An abstract class is a class designed with implementation gaps for subclasses to fill in and is deliberately incomplete.

****What is the difference between Array and vector?**
Array is a set of related data type and static whereas vector is a growable array of objects and dynamic.

….END of the journey

Credit goes to-
1.Yasin Javed Chowdhury (the JVM)
2.Quazi kutub Mia
3.Jahirul Islam
&
All my friends of R-11/CCSL/J2EE