**Core Java**

Hash code is an unique id number allocated to an object by JVM. Through this hash code only the object is referenced.  
  
Hash code is not an unique number for an object.If two objects are equals(means ob1.equals(ob2))then these two objects return same hash code.so we have to implement hashcode() of a class in such way that if two objects are equals(that is compared by equal() of that class)then those two objects must return same hash code.  
  
hashcode() method returns is integer but on JVM it is stored as hexa only.

### **Language Fundamentals**

**Question: How many number of non-public class definitions can a source file have A source file can contain unlimited number of non-public class definitions List primitive data types, there size and there range (min, max)**

**Answer:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data Type** | **Bytes** | **bits** | **min** | **max** |
| boolean | - | 1 | - | - |
| char | 2 | 16 | 0 | 2^16-1 |
| byte | 1 | 8 | -2^7 | 2^7-1 |
| short | 2 | 16 | -2^15 | 2^15-1 |
| int | 4 | 32 | -2^31 | 2^31-1 |
| long | 8 | 64 | -2^63 | 2^63-1 |
| float | 4 | 32 | - | - |
| double | 8 | 64 | - | - |

**Question: What types of values does boolean variables take It only takes values true and false. Which primitive datatypes are signed.**

**Answer:** All except char and Boolean  
  
**Question: Is char type signed or unsigned**

**Answer:** char type is integral but unsigned. It range is 0 to 2^7-1

**Question: What forms an integral literal can be**

**Answer:** decimal, octal and hexadecimal, hence example it can be 28, 034 and 0x1c respectively   
  
**Question: What is the default value of Boolean**

**Answer:** False   
  
**Question: Why is the main method static**

**Answer:** So that it can be invoked without creating an instance of that class   
  
**Question: What is the difference between class variable, member variable and automatic(local) variable**

**Answer:** class variable is a static variable and does not belong to instance of class but rather shared across all the instances   
member variable belongs to a particular instance of class and can be called from any method of the class   
automatic or local variable is created on entry to a method and has only method scope   
  
**Question: When are static and non static variables of the class initialized**

**Answer:** The static variables are initialized when the class is loadedNon static variables are initialized just before the constructor is called   
  
**Question: When are automatic variable initialized**

**Answer:** Automatic variable have to be initialized explicitly   
  
**Question: How is an argument passed in java, by copy or by reference**

**Answer:** If the variable is primitive datatype then it is passed by copy.   
If the variable is an object then it is passed by reference   
  
**Question: What is garbage collection**

**Answer:** The runtime [system](http://www.roseindia.net/interviewquestions/corejava/integral-literal.shtml) keeps track of the memory that is allocated and is able to determine whether that memory is still useable. This work is usually done in background by a low-priority thread that is referred to as garbage collector. When the gc finds memory that is no longer accessible from any live thread it takes steps to release it back to the heap for reuse   
  
**Question: Does System.gc and Runtime.gc() guarantee garbage collection**

**Answer:** No

### **Operators and assignment**

**Question: What are different types of operators in java**

**Answer:** Uniary ++, --, +, -, |, ~, ()   
Arithmetic \*, /, %,+, -   
Shift <<, >>, >>>   
Comparison =, instanceof, = =,!=Bitwise &, ^, |Short Circuit &&, ||Ternary ?:Assignment =   
  
**Question: How does bitwise (~) operator work**

**Answer:** It converts all the 1 bits in a binary value to 0s and all the 0 bits to 1s, e.g 11110000 coverts to 00001111   
  
**Question: What is a modulo operator %**

**Answer:** This operator gives the value which is related to the remainder of a divisione.g x=7%4 gives remainder 3 as an answer   
  
**Question: Can shift operators be applied to float types.**

**Answer:** No, shift operators can be applied only to integer or long types   
  
**Question: What happens to the bits that fall off after shifting**

**Answer:** They are discarded   
  
**Question: What values of the bits are shifted in after the shift**

**Answer:** In case of signed left shift >> the new bits are set to zero But in case of signed right shift it takes the value of most significant bit before the shift, that is if the most significant bit before shift is 0 it will introduce 0, else if it is 1, it will introduce 1

### **Modifiers**

**Question: What are access modifiers**

**Answer:** These public, protected and private, these can be applied to class, variables, constructors and methods. But if you don't specify an access modifier then it is considered as Friendly   
  
**Question: Can protected or friendly features be accessed from different packages**

**Answer:** No when features are friendly or protected they can be accessed from all the classes in that package but not from classes in another package   
  
**Question: How can you access protected features from another package**

**Answer:** You can access protected features from other classes by subclassing the that class in another package, but this cannot be done for friendly features   
  
**Question: What are the rules for overriding**

**Answer:**

Private method can be overridden by private, friendly, protected or public methods   
Friendly method can be overridden by friendly, protected or public methods   
Protected method can be overridden by protected or public methods   
Public method can be overridden by public method   
  
**Question: Explain modifier final**

**Answer:** Final can be applied to classes, methods and variables and the features cannot be changed. Final class cannot be subclassed, methods cannot be overridden   
  
**Question: Can you change the reference of the final object**

**Answer:** No the reference cannot be change, but the data in that object can be changed   
  
**Question: Can abstract modifier be applied to a variable**

**Answer:** No it is applied only to class and methods   
  
**Question: Can abstract class be instantiated**

**Answer:** No abstract class cannot be instantiated i.e you cannot create a new object of this class

**Question: When does the compiler insist that the class must be abstract**

**Answer:**

If one or more methods of the class are abstract.   
If class inherits one or more abstract methods from the parent abstract class and no implementation is provided for that method   
If class implements an interface and provides no implementation for those methods   
  
**Question: How is abstract class different from final class**

**Answer:** Abstract class must be subclassed and final class cannot be subclassed   
  
**Question: Where can static modifiers be used**

**Answer:** They can be applied to variables, methods and even a block of code, static methods and variables are not associated with any instance of class   
  
**Question: When are the static variables loaded into the memory**

**Answer:** During the class [load time](http://www.roseindia.net/interviewquestions/corejava/modifiers.shtml)   
  
**Question: When are the non static variables loaded into the memory**

**Answer:** They are loaded just before the constructor is called   
  
**Question: How can you reference static variables**

**Answer:** Via reference to any instance of the class

[Computer](http://www.roseindia.net/interviewquestions/corejava/modifiers.shtml) comp = new Computer ();   
comp.harddisk where hardisk is a static variable   
comp.compute() where compute is a method

Via the class name

Computer.harddisk   
Computer.compute()

**Question: Can static method use non static features of there class**

**Answer:** No they are not allowed to use non static features of the class, they can only call static methods and can use static data   
  
**Question: What is static initializer code**

**Answer:** A class can have a block of initializer code that is simply surrounded by curly braces and labeled as static e.g.   
public class Demo{   
static int =10;   
static{   
[System](http://www.roseindia.net/interviewquestions/corejava/modifiers.shtml).out.println(“Hello world');   
}   
}   
  
And this code is executed exactly once at the time of class load   
  
Where is native modifier used It can refer only to methods and it indicates that the body of the method is to be found else where and it is usually written in non [java language](http://www.roseindia.net/interviewquestions/corejava/modifiers.shtml)   
  
**Question: What are transient variables**

**Answer:** A transient variable is not stored as part of objects persistent state and they cannot be final or static   
  
**Question: What is synchronized modifier used for**

**Answer:** It is used to control access of critical code in [multithreaded](http://www.roseindia.net/interviewquestions/corejava/modifiers.shtml) programs   
  
**Question: What are volatile variables**

**Answer:** It indicates that these variables can be modified asynchronously

### **Conversion Casting and Promotion**

**Question: What are wrapped** [**classes**](http://www.roseindia.net/interviewquestions/corejava/conversion-casting-promotion.shtml)

**Answer:** Wrapped classes are classes that allow primitive types to be accessed as objects.   
  
**Question: What are the four general cases for Conversion and Casting**

**Answer:**

Conversion of primitives   
Casting of primitives   
Conversion of object references   
Casting of object references   
  
**Question: When can conversion happen**

**Answer:**

It can happen during

Assignment   
Method call   
[Arithmetic](http://www.roseindia.net/interviewquestions/corejava/conversion-casting-promotion.shtml) promotion

**Question: What are the rules for primitive assignment and method call conversion**

**Answer:**

A boolean can not be converted to any other type   
A non Boolean can be converted to another non boolean type, if the conversion is widening conversion   
A non Boolean cannot be converted to another non boolean type, if the conversion is narrowing conversion   
See figure below for simplicity   
assignment  
**Question: What are the rules for primitive arithmetic promotion conversion**

**Answer:**

For Unary operators :

If operant is byte, short or a char it is converted to an int   
If it is any other type it is not converted

For binary operands :

If one of the operands is double, the other operand is converted to double   
Else If one of the operands is float, the other operand is converted to float   
Else If one of the operands is long, the other operand is converted to long   
Else both the operands are converted to int

**Question: What are the rules for casting primitive types**

**Answer:**

You can cast any non Boolean type to any other non boolean type   
You cannot cast a boolean to any other type; you cannot cast any other type to a boolean   
  
**Question: What are the rules for object reference assignment and method call conversion**

**Answer:** An interface type can only be converted to an interface type or to object. If the new type is an [interface](http://www.roseindia.net/interviewquestions/corejava/conversion-casting-promotion.shtml), it must be a superinterface of the old type. A class type can be converted to a class type or to an interface type. If converting to a class type the new type should be superclass of the old type. If converting to an interface type new type the old class must implement the interface. An array maybe converted to class object, to the interface cloneable, or to an array. Only an array of object references types may be converted to an array, and the old element type must be convertible to the new element.   
  
**Question: What are the rules for Object reference casting**

**Answer:** Casting from Old types to Newtypes   
Compile time rules

* When both Oldtypes and Newtypes are classes, one should be subclass of the other
* When both Oldtype ad Newtype are arrays, both arrays must contain reference types (not primitive), and it must be legal to cast an element of Oldtype to an element of Newtype
* You can always cast between an interface and a non-final object

Runtime rules

* If Newtype is a class. The class of the expression being converted must be Newtype or must inherit from Newtype
* If NewType is an interface, the class of the expression being converted must implement Newtype

### **Flow Control and exception**

**Question: What is the difference between while and do while loop**

**Answer:** Do while loop walways executes the body of the loop at least once, since the test is performed at the end of the body   
  
**Question: When do you use continue and when do you use break statements**

**Answer:** When continue statement is applied it prematurely completes the iteration of a loop.   
When break statement is applied it causes the entire loop to be abandoned.   
  
**Question: What is the base class from which all exceptions are subclasses**

**Answer:** All exceptions are subclasses of a class called [java](http://www.roseindia.net/interviewquestions/corejava/flow-control-exception.shtml).lang.Throwable   
  
**Question: How do you intercept and thereby control exceptions**

**Answer:** We can do this by using try/catch/finally blocks   
You place the normal processing code in try block   
You put the code to deal with exceptions that might arise in try block in catch block   
Code that must be executed no matter what happens must be place in finally block   
  
**Question: When do we say an exception is handled**

**Answer:** When an exception is thrown in a try block and is caught by a matching catch block, the exception is considered to have been handled   
  
**Question: When do we say an exception is not handled**

**Answer:** There is no catch block that names either the class of exception that has been thrown or a class of exception that is a parent class of the one that has been thrown, then the exception is considered to be unhandled, in such condition the execution leaves the method directly as if no try has been executed   
  
**Question: In what sequence does the finally block gets executed**

**Answer:** If you put finally after a try block without a matching catch block then it will be executed after the try block   
If it is placed after the catch block and there is no exception then also it will be executed after the try block   
If there is an exception and it is handled by the catch block then it will be executed after the catch block   
  
**Question: What can prevent the execution of the code in finally block**

**Answer:**

* The death of thread
* Use of [system](http://www.roseindia.net/interviewquestions/corejava/flow-control-exception.shtml).exit()
* Turning off the power to [CPU](http://www.roseindia.net/interviewquestions/corejava/flow-control-exception.shtml)
* An exception arising in the finally block itself

What are the rules for catching multiple exceptions   
A more specific catch block must precede a more general one in the source, else it gives compilation error   
Only one catch block, that is first applicable one, will be executed   
  
**Question: What does throws statement declaration in a method indicate**

**Answer:** This indicates that the method throws some exception and the caller method should take care of handling it   
  
**Question: What are checked exception**

**Answer:** Checked exceptions are exceptions that arise in a correct program, typically due to user mistakes like entering wrong data or I/O problems   
  
**Question: What are runtime exceptions**

**Answer:** Runtime exceptions are due to [programming](http://www.roseindia.net/interviewquestions/corejava/flow-control-exception.shtml) bugs like out of bond arrays or null pointer exceptions.   
  
**Question: What is difference between Exception and errors**

**Answer:** Errors are usually compile time and exceptions can be runtime or checked   
  
**Question: How will you handle the checked exceptions**

**Answer:** You can provide a try/catch block to handle it. OR Make sure method declaration includes a throws clause that informs the calling method an exception might be thrown from this particular method. When you extend a class and override a method, can this [new method](http://www.roseindia.net/interviewquestions/corejava/flow-control-exception.shtml) throw exceptions other than those that were declared by the original method. No it cannot throw, except for the subclasses of those exceptions.   
  
**Question: Is it legal for the extending class which overrides a method which throws an exception, not o throw in the overridden class**

**Answer:** Yes it is perfectly legal   
  
**Question: Explain the user defined Exceptions?**

**Answer:** User defined Exceptions are the separate Exception classes defined by the user for specific purposed. An user defined can created by simply sub-classing it to the Exception class. This allows custom exceptions to be generated (using throw) and caught in the same way as normal exceptions.

**Example:**  
            
class myCustomException extends Exception {   
// The class simply has to exist to be an exception   
}

OOPS

**.What are the principle concepts of OOPS?**

There are four principle concepts upon which object oriented design and programming rest. They are:

* Abstraction
* Polymorphism
* Inheritance
* Encapsulation (i.e. easily remembered as A-PIE).

**2.What is Abstraction?**

Abstraction refers to the act of representing essential features without including the background details or explanations.

**3.What is Encapsulation?**

Encapsulation is a technique used for hiding the properties and behaviors of an object and allowing outside access only as appropriate. It prevents other objects from directly altering or accessing the properties or methods of the encapsulated object.

**4.What is the difference between abstraction and encapsulation?**

* **Abstraction** focuses on the outside view of an object (i.e. the interface) **Encapsulation** (information hiding) prevents clients from seeing it’s inside view, where the behavior of the abstraction is implemented.
* **Abstraction** solves the problem in the design side while **Encapsulation** is the Implementation.
* **Encapsulation** is the deliverables of Abstraction. Encapsulation barely talks about grouping up your abstraction to suit the developer needs.

**5.What is Inheritance?**

* Inheritance is the process by which objects of one class acquire the properties of objects of another class.
* A class that is inherited is called a superclass.
* The class that does the inheriting is called a subclass.
* Inheritance is done by using the [keyword](http://www.developersbook.com/corejava/interview-questions/corejava-interview-questions-faqs.php) extends.
* The two most common reasons to use inheritance are:
  + To promote code reuse
  + To use polymorphism

**6.What is Polymorphism?**

Polymorphism is briefly described as "one interface, many implementations." Polymorphism is a characteristic of being able to assign a different meaning or usage to something in different contexts - specifically, to allow an entity such as a variable, a function, or an object to have more than one form.

**7.How does Java implement polymorphism?**

(Inheritance, Overloading and Overriding are used to achieve Polymorphism in java).  
Polymorphism manifests itself in Java in the form of multiple methods having the same name.

* In some cases, multiple methods have the same name, but different formal argument lists (overloaded methods).
* In other cases, multiple methods have the same name, same return type, and same formal argument list (overridden methods).

**8.Explain the different forms of Polymorphism.**

There are two types of polymorphism one is **Compile time polymorphism** and the other is run time polymorphism. Compile time polymorphism is method overloading. **Runtime time polymorphism** is done using inheritance and interface.  
**Note**: *From a practical programming viewpoint, polymorphism manifests itself in three distinct forms in Java:*

* *Method overloading*
* *Method overriding through inheritance*
* *Method overriding through the Java interface*

**9.What is runtime polymorphism or dynamic method dispatch?**

In Java, runtime polymorphism or dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime rather than at compile-time. In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

**10.What is Dynamic Binding?**

Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic binding (also known as late binding) means that the code associated with a given procedure call is not known until the time of the call at run-time. It is associated with polymorphism and inheritance.

**11.What is method overloading?**

Method Overloading means to have two or more methods with same name in the same class with different arguments. The benefit of method overloading is that it allows you to implement methods that support the same semantic [operation](http://www.developersbook.com/corejava/interview-questions/corejava-interview-questions-faqs.php) but differ by argument number or type.  
**Note**:

* *Overloaded methods MUST change the argument list*
* *Overloaded methods CAN change the return type*
* *Overloaded methods CAN change the access modifier*
* *Overloaded methods CAN declare new or broader checked exceptions*
* *A method can be overloaded in the same class or in a subclass*

**12.What is method overriding?**

Method overriding occurs when sub class declares a method that has the same type arguments as a method declared by one of its superclass. The key benefit of overriding is the ability to define behavior that’s specific to a particular subclass type.  
**Note**:

* *The overriding method cannot have a more restrictive access modifier than the method being overridden (Ex: You can’t override a method marked public and make it protected).*
* *You cannot override a method marked final*
* *You cannot override a method marked static*

**13.What are the differences between method overloading and method overriding?**

|  |  |  |
| --- | --- | --- |
|  | **Overloaded Method** | **Overridden Method** |
| **Arguments** | Must change | Must not change |
| **Return type** | Can change | Can’t change except for covariant returns |
| **Exceptions** | Can change | Can reduce or eliminate. Must not throw new or broader checked exceptions |
| **Access** | Can change | Must not make more restrictive (can be less restrictive) |
| **Invocation** | Reference type determines which overloaded version is selected. Happens at compile time. | Object type determines which method is selected. Happens at runtime. |

**14.Can overloaded methods be override too?**

Yes, derived classes still can override the overloaded methods. Polymorphism can still happen. [Compiler](http://www.developersbook.com/corejava/interview-questions/corejava-interview-questions-faqs.php) will not binding the method calls since it is overloaded, because it might be overridden now or in the future.

**15.Is it possible to override the main method?**

NO, because main is a static method. A static method can't be overridden in Java.

**16.How to invoke a superclass version of an Overridden method?**

To invoke a superclass method that has been overridden in a subclass, you must either call the method directly through a superclass instance, or use the super prefix in the subclass itself. From the point of the view of the subclass, the super prefix provides an explicit reference to the superclass' implementation of the method.

// From subclass

super.overriddenMethod();

**17.What is super?**

super is a keyword which is used to access the method or member variables from the superclass. If a method hides one of the member variables in its superclass, the method can refer to the hidden variable through the use of the super keyword. In the same way, if a method overrides one of the methods in its superclass, the method can invoke the overridden method through the use of the super keyword.   
**Note**:

* *You can only go back one level.*
* *In the constructor, if you use super(), it must be the very first code, and you cannot access any this.xxx variables or methods to compute its parameters.*

**18.How do you prevent a method from being overridden?**

To prevent a specific method from being overridden in a subclass, use the final modifier on the method declaration, which means "this is the final implementation of this method", the end of its inheritance hierarchy.

public final void exampleMethod() {  
                         // Method statements  
                         }

**19.What is an Interface?**

An interface is a description of a set of methods that conforming implementing classes must have.  
**Note**:

* *You can’t mark an interface as final.*
* *Interface variables must be static.*
* *An Interface cannot extend anything but another interfaces.*

**20.Can we instantiate an interface?**

You can’t instantiate an interface directly, but you can instantiate a class that implements an interface.

**21.Can we create an object for an interface?**

Yes, it is always necessary to create an object implementation for an interface. Interfaces cannot be instantiated in their own right, so you must write a class that implements the interface and fulfill all the methods defined in it.

**22.Do interfaces have member variables?**

Interfaces may have member variables, but these are implicitly public, static, and final- in other words, interfaces can declare only constants, not instance variables that are available to all implementations and may be used as key references for method arguments for example.

**23.What modifiers are allowed for methods in an Interface?**

Only public and abstract modifiers are allowed for methods in interfaces.

**24.What is a marker interface?**

Marker interfaces are those which do not declare any required methods, but signify their compatibility with certain operations. The java.io.Serializable interface and Cloneable are typical marker interfaces. These do not contain any methods, but classes must implement this interface in order to be serialized and de-serialized.

**25.What is an abstract class?**

Abstract classes are classes that contain one or more abstract methods. An abstract method is a method that is declared, but contains no implementation.   
**Note**:

* *If even a single method is abstract, the whole class must be declared abstract.*
* *Abstract classes may not be instantiated, and require subclasses to provide implementations for the abstract methods.*
* *You can’t mark a class as both abstract and final.*

**26.Can we instantiate an abstract class?**

An abstract class can never be instantiated. Its sole purpose is to be extended (subclassed).

**27.What are the differences between Interface and Abstract class?**

|  |  |
| --- | --- |
| **Abstract Class** | **Interfaces** |
| An abstract class can provide complete, default code and/or just the details that have to be overridden. | An interface cannot provide any code at all,just the signature. |
| In case of abstract class, a class may extend only one abstract class. | A Class may implement several interfaces. |
| An abstract class can have non-abstract methods. | All methods of an Interface are abstract. |
| An abstract class can have instance variables. | An Interface cannot have instance variables. |
| An abstract class can have any visibility: public, private, protected. | An Interface visibility must be public (or) none. |
| If we add a new method to an abstract class then we have the option of providing default implementation and therefore all the existing code might work properly. | If we add a new method to an Interface then we have to track down all the implementations of the interface and define implementation for the new method. |
| An abstract class can contain constructors . | An Interface cannot contain constructors . |
| Abstract classes are fast. | Interfaces are slow as it requires extra indirection to find corresponding method in the actual class. |

**28.When should I use abstract classes and when should I use interfaces?**

**Use Interfaces when…**

* You see that something in your design will change frequently.
* If various implementations only share method signatures then it is better to use Interfaces.
* you need some classes to use some methods which you don't want to be included in the class, then you go for the interface, which makes it easy to just implement and make use of the methods defined in the interface.

**Use Abstract Class when…**

* If various implementations are of the same kind and use common behavior or status then abstract class is better to use.
* When you want to provide a generalized form of abstraction and leave the implementation task with the inheriting subclass.
* Abstract classes are an excellent way to create planned inheritance hierarchies. They're also a good choice for nonleaf classes in class hierarchies.

**29.When you declare a method as abstract, can other nonabstract methods access it?**

Yes, other nonabstract methods can access a method that you declare as abstract.

**30.Can there be an abstract class with no abstract methods in it?**

Yes, there can be an abstract class without abstract methods.

**31.What is Constructor?**

* A constructor is a special method whose task is to initialize the object of its class.
* It is special because its name is the **same as the class name**.
* They do not have return types, not even **void** and therefore they cannot return values.
* They **cannot be inherited**, though a [derived class](http://www.developersbook.com/corejava/interview-questions/corejava-interview-questions-faqs-2.php) can call the base class constructor.
* Constructor is invoked whenever an object of its associated class is created.

**32.How does the Java default constructor be provided?**

If a class defined by the code does **not** have any constructor, [compiler](http://www.developersbook.com/corejava/interview-questions/corejava-interview-questions-faqs-2.php) will automatically provide one no-parameter-constructor (default-constructor) for the class in the byte code. The access modifier (public/private/etc.) of the default constructor is the same as the class itself.

**33.Can constructor be inherited?**

No, constructor cannot be inherited, though a derived class can call the base class constructor.

**34.What are the differences between Contructors and Methods?**

|  |  |  |
| --- | --- | --- |
|  | **Constructors** | **Methods** |
| **Purpose** | Create an instance of a class | Group Java statements |
| **Modifiers** | Cannot be *abstract, final, native, static*, or *synchronized* | Can be *abstract, final, native, static*, or *synchronized* |
| **Return Type** | No return type, not even void | Void or a valid return type |
| **Name** | Same name as the class (first letter is capitalized by convention) -- usually a noun | Any name except the class. Method names begin with a lowercase letter by convention -- usually the name of an action |
| ***this*** | Refers to another constructor in the same class. If used, it must be the first line of the constructor | Refers to an instance of the owning class. Cannot be used by static methods. |
| ***super*** | Calls the constructor of the parent class. If used, must be the first line of the constructor | Calls an overridden method in the parent class |
| **Inheritance** | Constructors are not inherited | Methods are inherited |

**35.How are this() and super() used with constructors?**

* Constructors use *this* to refer to another constructor in the same class with a different parameter list.
* Constructors use *super* to invoke the superclass's constructor. If a constructor uses *super*, it must use it in the first line; otherwise, the compiler will complain.

**36.What are the differences between Class Methods and Instance Methods?**

|  |  |
| --- | --- |
| **Class Methods** | **Instance Methods** |
| Class methods are methods which are declared as static. The method can be called without creating an instance of the class | Instance methods on the other hand require an instance of the class to exist before they can be called, so an instance of a class needs to be created by using the new keyword. Instance methods operate on specific instances of classes. |
| Class methods can only operate on class members and not on instance members as class methods are unaware of instance members. | Instance methods of the class can also not be called from within a class method unless they are being called on an instance of that class. |
| Class methods are methods which are declared as static. The method can be called without creating an  instance of the class. | Instance methods are not declared as static. |

**37.How are this() and super() used with constructors?**

* Constructors use *this* to refer to another constructor in the same class with a different parameter list.
* Constructors use *super* to invoke the superclass's constructor. If a constructor uses super, it must use it in the first line; otherwise, the compiler will complain.

**38.What are Access Specifiers?**

One of the techniques in object-oriented programming is *encapsulation*. It concerns the hiding of data in a class and making this class available only through methods. Java allows you to control access to classes, methods, and fields via so-called *access specifiers*..

**39.What are Access Specifiers available in Java?**

Java offers four access specifiers, listed below in decreasing accessibility:

* **Public**- *public* classes, methods, and fields can be accessed from everywhere.
* **Protected**- *protected* methods and fields can only be accessed within the same class to which the methods and fields belong, within its subclasses, and within classes of the same package.
* **Default(no specifier)-** If you do not set access to specific level, then such a class, method, or field will be accessible from inside the same package to which the class, method, or field belongs, but not from outside this package.
* **Private**- *private* methods and fields can only be accessed within the same class to which the methods and fields belong. *private* methods and fields are not visible within subclasses and are not inherited by subclasses.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Situation** | **public** | **protected** | **default** | **private** |
| Accessible to class   from same package? | yes | yes | yes | no |
| Accessible to class   from different package? | yes | no, *unless it is a subclass* | no | no |

**40.What is final modifier?**

The final modifier keyword makes that the programmer cannot change the value anymore. The actual meaning depends on whether it is applied to a class, a variable, or a method.

* ***final* Classes**- A final class cannot have subclasses.
* ***final* Variables**- A final variable cannot be changed once it is initialized.
* ***final* Methods**- A final method cannot be overridden by subclasses.

**41.What are the uses of final method?**

There are two reasons for marking a method as final:

* Disallowing subclasses to change the meaning of the method.
* Increasing efficiency by allowing the compiler to turn calls to the method into inline [Java code](http://www.developersbook.com/corejava/interview-questions/corejava-interview-questions-faqs-2.php).

**42.What is static block?**

Static block which exactly executed exactly once when the class is first loaded into JVM. Before going to the main method the static block will execute.

**43.What are static variables?**

Variables that have only one copy per class are known as static variables. They are not attached to a particular instance of a class but rather belong to a class as a whole. They are declared by using the static keyword as a modifier.

static type varIdentifier;

where, the name of the variable is varIdentifier and its data type is specified by type.  
**Note**: Static variables that are not explicitly initialized in the code are automatically initialized with a default value. The default value depends on the data type of the variables.

**44.What is the difference between static and non-static variables?**

A static variable is associated with the class as a whole rather than with specific instances of a class. Non-static variables take on unique values with each object instance.

**45.What are static methods?**

Methods declared with the keyword static as modifier are called static methods or class methods. They are so called because they affect a class as a whole, not a particular instance of the class. Static methods are always invoked without reference to a particular instance of a class.  
**Note**:The use of a static method suffers from the following restrictions:

* *A static method can only call other static methods.*
* *A static method must only access static data.*
* *A static method* ***cannot*** *reference to the current object using keywords super or this.*

Exception H*andling*

**1.What is an exception?**

An *exception* is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.

**2.What is error?**

An Error indicates that a non-recoverable condition has occurred that should not be caught. Error, a subclass of Throwable, is intended for drastic problems, such as OutOfMemoryError, which would be reported by the JVM itself.

**3.Which is superclass of Exception?**

**"Throwable"**, the parent class of all exception related classes.

Q.What are the different scopes for Java variables?

\

The scope of a Java variable is determined by the context in which the variable is declared. Thus a java variable can have one of the three scopes at any given point in time.   
1. Instance : - These are typical object level variables, they are initialized to default values at the time of creation of object, and remain accessible as long as the object accessible.   
2. Local : - These are the variables that are defined within a method. They remain accessbile only during the course of method excecution. When the method finishes execution, these variables fall out of scope.   
3. Static: - These are the class level variables. They are initialized when the class is loaded in JVM for the first time and remain there as long as the class remains loaded. They are not tied to any particular object instance

**Question:** What is OOPS?  
**Answer:** OOP is the common abbreviation for Object-Oriented Programming.  

**Question:** Describe the principles of OOPS.  
**Answer:** There are three main principals of oops which are called Polymorphism, Inheritance and Encapsulation.  

**Question:** Explain the Encapsulation principle.  
**Answer:** Encapsulation is a process of binding or wrapping the data and the codes that operates on the data into a single entity. This keeps the data safe from outside interface and misuse. One way to think about encapsulation is as a protective wrapper that prevents code and data from being arbitrarily accessed by other code defined outside the wrapper.  

**Question:** Explain the Inheritance principle.  
**Answer:** Inheritance is the process by which one object acquires the properties of another object.  

**Question:** Explain the Polymorphism principle.  
**Answer:** The meaning of Polymorphism is something like one name many forms. Polymorphism enables one entity to be used as as general category for different types of actions. The specific action is determined by the exact nature of the situation. The concept of polymorphism can be explained as "one interface, multiple methods".  

**Question:** Explain the different forms of Polymorphism.  
**Answer:** From a practical programming viewpoint, polymorphism exists in three distinct forms in Java:

* Method overloading
* Method overriding through inheritance
* Method overriding through the Java interface

**Question:** What are Access Specifiers available in Java?  
**Answer:** Access specifiers are keywords that determines the type of access to the member of a class. These are:

* Public
* Protected
* Private
* Defaults

**Question:** Describe the wrapper classes in Java.  
**Answer:** Wrapper class is wrapper around a primitive data type. An instance of a wrapper class contains, or wraps, a primitive value of the corresponding type.

Following table lists the primitive types and the corresponding wrapper classes:

|  |  |
| --- | --- |
| **Primitive** | **Wrapper** |
| **boolean** | **java.lang.Boolean** |
| **byte** | **java.lang.Byte** |
| **char** | **java.lang.Character** |
| **double** | **java.lang.Double** |
| **float** | **java.lang.Float** |
| **int** | **java.lang.Integer** |
| **long** | **java.lang.Long** |
| **short** | **java.lang.Short** |
| **void** | **java.lang.Void** |

**Question:** Read the following program:

public class test {  
public static void main(String [] args) {  
    int x = 3;  
    int y = 1;  
   if (x = y)  
     System.out.println("Not equal");  
  else  
    System.out.println("Equal");  
 }  
}

What is the result?  
   A. The output is “Equal”  
   B. The output in “Not Equal”  
   C. An error at " if (x = y)" causes compilation to fall.  
   D. The program executes but no output is show on console.  
**Answer: C  
  
Question:** what is the class variables ?  
**Answer:** When we create a number of objects of the same class, then each object will share a common copy of variables. That means that there is only one copy per class, no matter how many objects are created from it. Class variables or static variables are declared with the static keyword in a class, but mind it that it should be declared outside outside a class. These variables are stored in static memory. Class variables are mostly used for constants, variable that never change its initial value. Static variables are always called by the class name. This variable is created when the program starts i.e. it is created before the instance is created of class by using new operator and gets destroyed when the programs stops. The scope of the class variable is same a instance variable. The class variable can be defined anywhere at class level with the keyword static. It initial value is same as instance variable. When the class variable is defined as int then it's initial value is by default zero, when declared boolean its default value is false and null for object references. Class variables are associated with the class, rather than with any object.

**Question:** What is the difference between the instanceof and getclass, these two are same or not ?  
**Answer:** instanceof is a operator, not a function while getClass is a method of java.lang.Object class. Consider a condition where we use   
if(o.getClass().getName().equals("java.lang.Math")){ }  
This method only checks if the classname we have passed is equal to java.lang.Math. The class java.lang.Math is loaded by the bootstrap ClassLoader. This class is an abstract class.This class loader is responsible for loading classes. Every Class object contains a reference to the ClassLoader that defines. getClass() method returns the runtime class of an object. It fetches the java instance of the given fully qualified type name. The code we have written is not necessary, because we should not compare getClass.getName(). The reason behind it is that if the two different class loaders load the same class but for the JVM, it will consider both classes as different classes so, we can't compare their names. It can only gives the implementing class but can't compare a interface, but instanceof operator can.   
The instanceof operator compares an object to a specified type. We can use it to test if an object is an instance of a class, an instance of a subclass, or an instance of a class that implements a particular interface. We should try to use instanceof operator in place of getClass() method. Remember instanceof opeator and getClass are not same. Try this example, it will help you to better understand the difference between the two.   
Interface one{  
}  
  
Class Two implements one {  
}  
Class Three implements one {  
}  
  
public class Test {  
public static void main(String args[]) {  
one test1 = new Two();  
one test2 = new Three();  
System.out.println(test1 instanceof one); //true  
System.out.println(test2 instanceof one); //true  
System.out.println(Test.getClass().equals(test2.getClass())); //false  
}  
}

**Question: How could Java classes direct program messages to the system console, but error messages, say to a file?**

**Answer:** The class [System](http://www.roseindia.net/interviewquestions/corejava/Java-classes-direct-program-messages.shtml) has a variable *out* that represents the standard output, and the variable *err* that represents the standard error device. By default, they both point at the system console. This how the standard output could be re-directed:

Stream st = new Stream(new FileOutputStream("output.txt")); System.setErr(st); System.setOut(st);

**Question: What's the difference between an interface and an abstract class?**

**Answer:** An abstract class may contain code in method bodies, which is not allowed in an interface. With abstract classes, you have to inherit your class from it and [Java](http://www.roseindia.net/interviewquestions/corejava/Java-classes-direct-program-messages.shtml) does not allow multiple inheritance. On the other hand, you can implement multiple interfaces in your class.

**Question: Why would you use a synchronized block vs. synchronized method?**

**Answer:** Synchronized blocks place locks for shorter periods than synchronized methods.

**Question: Explain the usage of the keyword transient?**

**Answer:** This keyword indicates that the value of this member variable does not have to be serialized with the object. When the class will be de-serialized, this variable will be initialized with a [default value](http://www.roseindia.net/interviewquestions/corejava/Java-classes-direct-program-messages.shtml) of its data type (i.e. zero for integers). **Question: How can you force garbage collection?**

**Answer:** You can't force GC, but could request it by calling System.gc(). JVM does not guarantee that GC will be started immediately.

**Question: How do you know if an explicit object casting is needed?**

**Answer:** If you assign a superclass object to a variable of a subclass's data type, you need to do explicit casting. For example:

Object a; Customer b; b = (Customer) a;

When you assign a subclass to a variable having a supeclass type, the casting is performed automatically.

**Question: Can you write a Java class that could be used both as an applet as well as an application?**

**Answer:** Yes. Add a main() method to the applet.

**Question: What's the difference between constructors and other methods?**

**Answer:** Constructors must have the same name as the class and can not return a value. They are only called once while regular methods could be called many times.

**Question: Can you call one constructor from another if a class has multiple constructors**

Answer: Yes. Use this() syntax.

**Question: Explain the usage of Java packages.**

Answer: 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](http://www.roseindia.net/interviewquestions/corejava/Java-classes-direct-program-messages.shtml) from being used by the non-authorized classes.

**Question: If a class is located in a package, what do you need to change in the OS environment to be able to use it?**

**Answer:** You need to add a directory or a jar file that contains the package directories to the CLASSPATH [environment variable](http://www.roseindia.net/interviewquestions/corejava/Java-classes-direct-program-messages.shtml). Let's say a class Employee belongs to a package com.xyz.hr; and is located in the file c:\dev\com\xyz\hr\Employee.java. In this case, you'd need to add c:\dev to the variable CLASSPATH. If this class contains the method main(), you could test it from a command prompt window as follows:

c:\>java com.xyz.hr.Employee

**Question: What's the difference between J2SDK 1.5 and J2SDK 5.0?**

**Answer:**  There's no difference, Sun Microsystems just re-branded this version.   
  
**Question:** What would you use to compare two String variables - the operator == or the method equals()?

**Answer:**  I'd use the method equals() to compare the values of the Strings and the == to check if two variables point at the same instance of a String object.

**Question: Does it matter in what order catch statements for FileNotFoundException and IOExceptipon are written?**

**Answer:** Yes, it does. The FileNoFoundException is inherited from the IOException. Exception's subclasses have to be caught first.

**Question: Can an inner class declared inside of a method access local variables of this method?**

**Answer:** It's possible if these variables are final.

**Question: What can go wrong if you replace && with & in the following code: String a=null; if (a!=null && a.length()>10) {...}**

**Answer:** A single ampersand here would lead to a NullPointerException.

**Question:** **What's the main difference between a Vector and an ArrayList**

**Answer:** [Java](http://www.roseindia.net/interviewquestions/corejava/IOExceptipon.shtml) Vector class is internally synchronized and ArrayList is not. **Question: When should the method invokeLater()be used?**

**Answer:** This method is used to ensure that Swing components are updated through the event-dispatching thread.

**Question: How can a subclass call a method or a constructor defined in a superclass?**

**Answer:** Use the following syntax: super.myMethod(); To call a constructor of the superclass, just write super(); in the first line of the subclass's constructor.

**For senior-level developers:**

**Question: What's the difference between a queue and a stack?**

**Answer:** Stacks works by last-in-first-out rule (LIFO), while queues use the FIFO rule

**Question:** **You can create an abstract class that contains only abstract methods. On the other hand, you can create an interface that declares the same methods. So can you use abstract classes instead of interfaces?**

**Answer:** Sometimes. But your class may be a descendent of another class and in this case the [interface](http://www.roseindia.net/interviewquestions/corejava/IOExceptipon.shtml) is your only option.

**Question: What comes to mind when you hear about a young generation in Java?**

**Answer:** Garbage collection.

**Question: What comes to mind when someone mentions a shallow copy in Java?**

**Answer:** Object cloning.

**Question: If you're overriding the method equals() of an object, which other method you might also consider?**

**Answer:** hashCode()

**Question: How would you make a copy of an entire Java object with its state?**

**Answer:** Have this class implement Cloneable interface and call its method clone().

**Question: How can you minimize the need of garbage collection and make the memory use more effective?**

**Answer:** Use object pooling and weak object references.

**Question: There are two classes: A and B. The class B need to inform a class A when some important event has happened. What Java technique would you use to implement it?**

**Answer:** If these classes are threads I'd consider notify() or notifyAll(). For regular classes you can use the Observer interface.

**Question: What access level do you need to specify in the class declaration to ensure that only classes from the same directory can access it?**

**Answer:** You do not need to specify any access level, and Java will use a default package access level .

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Question:** When you declare a method as abstract method ?  **Answer:** When i want child class to implement the behavior of the method.  **Question:** Can I call a abstract method from a non abstract method ?  **Answer:** Yes, We can call a abstract method from a Non abstract method in a [Java](http://www.roseindia.net/interviewquestions/corejava/abstract-method.shtml) abstract class  **Question:** What is the difference between an Abstract class and [Interface](http://www.roseindia.net/interviewquestions/corejava/abstract-method.shtml) in Java ? or can you explain when you use Abstract classes ?  **Answer:** Abstract classes let you define some behaviors; they force your subclasses to provide others. These abstract classes will provide the basic funcationality of your applicatoin, child class which inherited this class will provide the funtionality of the abstract methods in abstract class. When [base class](http://www.roseindia.net/interviewquestions/corejava/abstract-method.shtml) calls this method, Java calls the method defined by the child class.   * An Interface can only declare constants and instance methods, but cannot implement default behavior. * Interfaces provide a form of multiple inheritance. A class can extend only one other class. * Interfaces are limited to public methods and constants with no implementation. Abstract classes can have a partial implementation, protected parts, static methods, etc. * A Class may implement several interfaces. But in case of abstract class, a class may extend only one abstract class. * Interfaces are slow as it requires extra indirection to find corresponding method in the actual class. Abstract classes are fast.   **Question:** What are the field/method access levels (specifiers) and class access levels ?  **Answer:** Each field and method has an access level:   * private: accessible only in this class * (package): accessible only in this package * protected: accessible only in this package and in all subclasses of this class * public: accessible everywhere this class is available   **Similarly, each class has one of two possible access levels:**   * (package): class objects can only be declared and manipulated by code in this package * public: class objects can be declared and manipulated by code in any package   **Question:** What are the static fields & static Methods ?  **Answer:** If a field or method defined as a static, there is only one copy for entire class, rather than one copy for each instance of class. static method cannot accecss non-static field or call non-static method   Example [Java Code](http://www.roseindia.net/interviewquestions/corejava/checked-unchecked-exceptions.shtml)   static int counter = 0;   A public static field or method can be accessed from outside the class using either the usual notation:   Java-class-object.field-or-method-name   or using the class name instead of the name of the class object:   Java- class-name.field-or-method-name  **Question:** What are the Final fields & Final Methods ?  **Answer:** Fields and methods can also be declared final. A final method cannot be overridden in a subclass. A final field is like a constant: once it has been given a value, it cannot be assigned to again.   Java Code   private static final int MAXATTEMPTS = 10;  **Question:** Describe the wrapper classes in Java ?  **Answer:** Wrapper class is wrapper around a primitive data type. An instance of a wrapper class contains, or wraps, a primitive value of the corresponding type.   Following table lists the primitive types and the corresponding wrapper classes:   |  |  | | --- | --- | | **Primitive** | **Wrapper** | | boolean | java.lang.Boolean | | byte | java.lang.Byte | | char | java.lang.Character | | double | java.lang.Double | | float | java.lang.Float | | int | java.lang.Integer | | long | java.lang.Long | | short | java.lang.Short | | void | java.lang.Void |   **Question:** What are different types of inner classes ?  **Answer:** Inner classes nest within other classes. A normal class is a direct member of a package. Inner classes, which became available with [Java](http://www.roseindia.net/interviewquestions/corejava/inner-classes.shtml) 1.1, are four types   * Static member classes * Member classes * Local classes * Anonymous classes   **Static member classes** - a static member class is a static member of a class. Like any other static method, a static member class has access to all static methods of the parent, or top-level, class.   **Member Classes** - a member class is also defined as a member of a class. Unlike the static variety, the member class is instance specific and has access to any and all methods and members, even the parent's this reference.   **Local Classes** - Local Classes declared within a block of code and these classes are visible only within the block.   **Anonymous Classes** - These type of classes does not have any name and its like a local class  Java Anonymous Class Example public class SomeGUI extends JFrame { ... button member declarations ... protected void buildGUI() { button1 = new JButton(); button2 = new JButton(); ... button1.addActionListener( new java.awt.event.ActionListener() <------ Anonymous Class { public void actionPerformed(java.awt.event.ActionEvent e) { // do something } } );  **Question:** What are the uses of [Serialization](http://www.roseindia.net/interviewquestions/corejava/inner-classes.shtml)?  **Answer:** In some types of applications you have to write the code to serialize objects, but in many cases serialization is performed behind the scenes by various server-side containers.   These are some of the typical uses of serialization:   * To persist data for future use. * To send data to a remote computer using such client/server Java [technologies](http://www.roseindia.net/interviewquestions/corejava/inner-classes.shtml) as RMI or socket programming. * To "flatten" an object into array of bytes in memory. * To exchange data between applets and servlets. * To store user session in Web applications. * To activate/passivate enterprise [java beans](http://www.roseindia.net/interviewquestions/corejava/inner-classes.shtml). * To send objects between the servers in a cluster.   **Question:** For concatenation of strings, which method is good, StringBuffer or String ?  **Answer:** StringBuffer is faster than String for concatenation.  **Question :** How you can force the garbage collection ?  **Answer :** Garbage collection automatic process and can't be forced. We can call garbage collector in Java by calling System.gc() and Runtime.gc(), JVM tries to recycle the unused objects, but there is no guarantee when all the objects will garbage collected.  **Question :** What are the field/method access levels (specifiers) and class access levels ?  **Answer:** Each field and method has an access level:   * private: accessible only in this class * (package): accessible only in this package * protected: accessible only in this package and in all subclasses of this class * public: accessible everywhere this class is available   Similarly, each class has one of two possible access levels:   * (package): class objects can only be declared and manipulated by code in this package   public: class objects can be declared and manipulated by code in any package  **Question:** What are the static fields & static Methods ?  **Answer:** If a field or method defined as a static, there is only one copy for entire class, rather than one copy for each instance of class. static method cannot accecss non-static field or call non-static method   Example [Java Code](http://www.roseindia.net/interviewquestions/corejava/static-fields-&-methods.shtml)   static int counter = 0;   A public static field or method can be accessed from outside the class using either the usual notation:   Java-class-object.field-or-method-name   or using the class name instead of the name of the class object:   Java- class-name.field-or-method-name  **Question:** What are the Final fields & Final Methods ?  **Answer:** Fields and methods can also be declared final. A final method cannot be overridden in a subclass. A final field is like a constant: once it has been given a value, it cannot be assigned to again.   Java Code   private static final int MAXATTEMPTS = 10;  **Question:** Describe the wrapper classes in [Java](http://www.roseindia.net/interviewquestions/corejava/static-fields-&-methods.shtml) ?  **Answer:** Wrapper class is wrapper around a primitive data type. An instance of a wrapper class contains, or wraps, a primitive value of the corresponding type.   Following table lists the primitive types and the corresponding wrapper [classes](http://www.roseindia.net/interviewquestions/corejava/static-fields-&-methods.shtml):   |  |  | | --- | --- | | **Primitive** | **Wrapper** | | boolean | java.lang.Boolean | | byte | java.lang.Byte | | char | java.lang.Character | | double | java.lang.Double | | float | java.lang.Float | | int | java.lang.Integer | | long | java.lang.Long | | short | java.lang.Short | | void | java.lang.Void |   **Question:** What are different types of inner classes ?  **Answer:** Inner classes nest within other classes. A normal class is a direct member of a package. Inner classes, which became available with Java 1.1, are four types   * Static member classes * Member classes * Local classes * Anonymous classes   **Static member classes** - a static member class is a static member of a class. Like any other static method, a static member class has access to all static methods of the parent, or top-level, class.   **Member Classes** - a member class is also defined as a member of a class. Unlike the static variety, the member class is instance specific and has access to any and all methods and members, even the parent's this reference.   **Local Classes** - Local Classes declared within a block of code and these classes are visible only within the block.   **Anonymous Classes** - These type of classes does not have any name and its like a local class  Java Anonymous Class Example public class SomeGUI extends JFrame { ... button member declarations ... protected void buildGUI() { button1 = new JButton(); button2 = new JButton(); ... button1.addActionListener( new java.awt.event.ActionListener() <------ Anonymous Class { public void actionPerformed(java.awt.event.ActionEvent e) { // do something } } );  **Question:** What are the uses of [Serialization](http://www.roseindia.net/interviewquestions/corejava/uses-serialization.shtml)?  **Answer:** In some types of applications you have to write the code to serialize objects, but in many cases serialization is performed behind the scenes by various server-side containers.   These are some of the typical uses of serialization:   * To persist data for future use. * To send data to a remote computer using such client/server Java [technologies](http://www.roseindia.net/interviewquestions/corejava/uses-serialization.shtml) as RMI or socket programming. * To "flatten" an object into array of bytes in memory. * To exchange data between applets and servlets. * To store user session in Web applications . * To activate/passivate enterprise [java beans](http://www.roseindia.net/interviewquestions/corejava/uses-serialization.shtml). * To send objects between the servers in a cluster.   **Question:** what is a collection ?  **Answer:** Collection is a group of objects. [java](http://www.roseindia.net/interviewquestions/corejava/uses-serialization.shtml).util package provides important types of collections. There are two fundamental types of collections they are Collection and Map. Collection types hold a group of objects, Eg. Lists and Sets where as Map types hold group of objects as key, value pairs Eg. HashMap and Hashtable.  **Question:** For concatenation of strings, which method is good, StringBuffer or String ?  **Answer:** StringBuffer is faster than String for concatenation.  **Question: What is an Iterator interface?**  **Answer:** The Iterator interface is used to step through the elements of a Collection.  **Question: What is the difference between the >> and >>> operators?**  **Answer:** The >> operator carries the sign bit when shifting right. The >>> zero-fills bits that have been shifted out.  **Question: Which method of the Component class is used to set the position and size of a component?**  **Answer:** setBounds()  **Question: How many bits are used to represent Unicode,** [**ASCII**](http://www.roseindia.net/interviewquestions/corejava/iterator-interface.shtml)**, UTF-16, and UTF-8 characters?**  **Answer:** Unicode requires 16 bits and ASCII require 7 bits. Although the ASCII character set uses only 7 bits, it is usually represented as 8 bits. UTF-8 represents characters using 8, 16, and 18 bit patterns. UTF-16 uses 16-bit and larger bit patterns.  **Question: Which java.util classes and interfaces support event handling?**  **Answer:** The EventObject class and the EventListener interface support event processing.  **Question: Is sizeof a** [**keyword**](http://www.roseindia.net/interviewquestions/corejava/iterator-interface.shtml)**?**  **Answer:** The sizeof operator is not a keyword.  **Question: What are wrapped classes?**  **Answer:** Wrapped classes are classes that allow primitive types to be accessed as objects.  **Question: Does garbage collection guarantee that a program will not run out of memory?**  **Answer:** Garbage collection does not guarantee that a program will not run out of memory. It is possible for programs to use up memory resources faster than they are garbage collected. It is also possible for programs to create objects that are not subject to garbage collection  **Question:** **What restrictions are placed on the location of a package statement within a source code file?**  **Answer:** A package statement must appear as the first line in a [source code](http://www.roseindia.net/interviewquestions/corejava/iterator-interface.shtml) file (excluding blank lines and comments).  **Question: Can an object's finalize() method be invoked while it is reachable?**  **Answer:** An object's finalize() method cannot be invoked by the garbage collector while the object is still reachable. However, an object's finalize() method may be invoked by other objects.  **Question: What is the immediate superclass of the Applet class?**  **Answer:** Panel  **Question: What is the difference between preemptive scheduling and time slicing?**  **Answer:** Under preemptive scheduling, the highest priority task executes until it enters the waiting or dead states or a higher priority task comes into existence. Under time slicing, a task executes for a predefined slice of time and then reenters the pool of ready tasks. The scheduler then determines which task should execute next, based on priority and other factors.  **Question: Name three Component subclasses that support painting.**  **Answer:** The Canvas, Frame, Panel, and [Applet](http://www.roseindia.net/interviewquestions/corejava/iterator-interface.shtml) classes support painting.  **Question: What value does readLine() return when it has reached the end of a file?**  **Answer:** The readLine() [method returns](http://www.roseindia.net/interviewquestions/corejava/iterator-interface.shtml) null when it has reached the end of a file.  **Question: What is the immediate superclass of the Dialog class?**  **Answer:** Window  **Question: What is clipping?**  **Answer:** Clipping is the process of confining paint [operations](http://www.roseindia.net/interviewquestions/corejava/clipping.shtml) to a limited area or shape.  **Question: What is a native method?**  **Answer:** A native method is a method that is implemented in a language other than [Java](http://www.roseindia.net/interviewquestions/corejava/clipping.shtml).  **Question: Can a for statement loop indefinitely?**  **Answer:** Yes, a for statement can loop indefinitely. For example, consider the following: for(;;) ;  **Question: What are order of precedence and associativity, and how are they used?**  **Answer:** Order of precedence determines the order in which operators are evaluated in expressions. Associatity determines whether an expression is evaluated left-to-right or right-to-left  **Question: To what value is a variable of the String type automatically initialized?**  **Answer:** The [default value](http://www.roseindia.net/interviewquestions/corejava/clipping.shtml) of an String type is null.  **Question: What is the catch or declare rule for method declarations?**  **Answer:** If a checked exception may be thrown within the body of a method, the method must either catch the exception or declare it in its throws clause.  **Question: What is the difference between a MenuItem and a CheckboxMenuItem?**  **Answer:** The CheckboxMenuItem class extends the MenuItem class to support a [menu item](http://www.roseindia.net/interviewquestions/corejava/clipping.shtml) that may be checked or unchecked.  **Question: What is a task's priority and how is it used in scheduling?**  **Answer:** A task's priority is an [integer value](http://www.roseindia.net/interviewquestions/corejava/clipping.shtml) that identifies the relative order in which it should be executed with respect to other tasks. The scheduler attempts to schedule higher priority tasks before lower priority tasks.  **Question: What class is the top of the AWT event hierarchy?**  **Answer:** The java.awt.AWTEvent class is the highest-level class in the AWT event-class hierarchy.  **Question: Can an anonymous class be declared as implementing an interface and extending a class?**  **Answer:** An anonymous class may implement an interface or extend a superclass, but may not be declared to do both.  **Question: What is the range of the short type?**  **Answer:** The range of the short type is -(2^15) to 2^15 - 1.  **Question: What is the range of the char type?**  **Answer:** The range of the char type is 0 to 2^16 - 1.  **Question: In which package are most of the AWT events that support the event-delegation model defined?**  **Answer:** Most of the AWT-related events of the event-delegation model are defined in the java.awt.event package. The AWTEvent class is defined in the java.awt package.  **Question: What is the immediate superclass of Menu?**  **Answer:** MenuItem  **Question: What is the purpose of finalization?**  **Answer:** The purpose of finalization is to give an unreachable object the opportunity to perform any cleanup processing before the object is garbage collected.  **Question: Which class is the immediate superclass of the MenuComponent class.**  **Answer:** Object  **Question: What invokes a thread's run() method?**  **Answer:** After a thread is started, via its start() method or that of the Thread class, the [JVM](http://www.roseindia.net/interviewquestions/corejava/threads-run-method.shtml) invokes the thread's run() method when the thread is initially executed.  **Question: What is the difference between the Boolean & operator and the && operator?**  **Answer:** If an expression involving the Boolean & operator is evaluated, both operands are evaluated. Then the & operator is applied to the operand. When an expression involving the && operator is evaluated, the first operand is evaluated. If the first operand returns a value of true then the second operand is evaluated. The && operator is then applied to the first and second operands. If the first operand evaluates to false, the evaluation of the second operand is skipped.  **Question: Name three subclasses of the Component class.**  **Answer:** Box.Filler, Button, Canvas, Checkbox, Choice, Container, Label, List, [Scrollbar](http://www.roseindia.net/interviewquestions/corejava/threads-run-method.shtml), or TextComponent  **Question: What is the GregorianCalendar class?**  **Answer:** The GregorianCalendar provides support for traditional Western calendars.  **Question: Which Container method is used to cause a container to be laid out and redisplayed?**  **Answer:** validate()  **Question: What is the purpose of the Runtime class?**  **Answer:** The purpose of the Runtime class is to provide access to the Java runtime [system](http://www.roseindia.net/interviewquestions/corejava/threads-run-method.shtml).  **Question: How many times may an object's finalize() method be invoked by the garbage collector?**  **Answer:** An object's finalize() method may only be invoked once by the garbage collector.  **Question: What is the purpose of the finally clause of a try-catch-finally statement?**  **Answer:** The finally clause is used to provide the capability to execute code no matter whether or not an exception is thrown or caught.  **Question: What is the argument type of a program's main() method?**  **Answer:** A program's main() method takes an argument of the String[] type.  **Question: Which Java operator is right associative?**  **Answer:** The = operator is right associative.  **Question: What is the Locale class?**  **Answer:** The Locale class is used to tailor program output to the conventions of a particular geographic, political, or cultural region.  **Question: Can a double value be cast to a byte?**  **Answer:** Yes, a double value can be cast to a byte.  **Question: What is the difference between a break statement and a continue statement?**  **Answer:** A break statement results in the termination of the statement to which it applies (switch, for, do, or while). A continue statement is used to end the current loop iteration and return control to the loop statement.  **Question: What must a class do to implement an interface?**  **Answer:** It must provide all of the methods in the interface and identify the interface in its implements clause.  **Question: Name two subclasses of the TextComponent class.**  **Answer:** TextField and TextArea  **Question: What is the advantage of the event-delegation model over the earlier event-inheritance model?**  **Answer:** The event-delegation model has two advantages over the event-inheritance model. First, it enables [event handling](http://www.roseindia.net/interviewquestions/corejava/threads-run-method.shtml) to be handled by objects other than the ones that generate the events (or their containers). This allows a clean separation  between a component's design and its use. The other advantage of the event-delegation model is that it performs much better in [applications](http://www.roseindia.net/interviewquestions/corejava/threads-run-method.shtml) where many events are generated. This performance improvement is due to the fact that the event-delegation model does not have to repeatedly process unhandled events, as is the case of the event-inheritance model.  **Question: Which containers may have a MenuBar?**  **Answer:** Frame  **Question: How are commas used in the intialization and iterationparts of a for statement?**  **Answer:** Commas are used to separate multiple statements within the initialization and iteration parts of a for statement.  **Question: What is an abstract method?**  **Answer:** An abstract method is a method whose implementation is deferred to a subclass.  **Question: How are** [**Java source code**](http://www.roseindia.net/interviewquestions/corejava/menu-bar.shtml) **files named?**  **Answer:** A Java source code file takes the name of a public class or [interface](http://www.roseindia.net/interviewquestions/corejava/menu-bar.shtml) that is defined within the file. A [source code](http://www.roseindia.net/interviewquestions/corejava/menu-bar.shtml) file may contain at most one public class or interface. If a public class or interface is defined within a source code file, then the source code file must take the name of the public class or interface. If no public class or interface is defined  within a source code file, then the file must take on a name that is different than its classes and interfaces. Source code files use the .[java](http://www.roseindia.net/interviewquestions/corejava/menu-bar.shtml) extension.  **Question: What is the relationship between the Canvas class and the Graphics class?**  **Answer:** A Canvas object provides access to a Graphics object via its paint() method.  **Question: What value does read() return when it has reached the end of a file?**  **Answer:** The read() method returns -1 when it has reached the end of a file.  **Question: Can a Byte object be cast to a double value?**  **Answer:** No, an object cannot be cast to a primitive value.  **Question: What is the difference between a static and a non-static inner class?**  **Answer:** A non-static inner class may have object instances that are associated with instances of the class's outer class. A static inner class does not have any object instances.  **Question: What is the difference between the String and StringBuffer classes?**  **Answer:** String objects are constants. StringBuffer objects are not.  **Question: If a variable is declared as private, where may the variable be accessed?**  **Answer:** A private variable may only be accessed within the class in which it is declared.  **Question: What is the Dictionary class?**  **Answer:** The Dictionary class provides the capability to store key-value pairs.  **Question: How are the elements of a BorderLayout organized?**  **Answer:** The elements of a BorderLayout are organized at the borders (North, South, East, and West) and the center of a container.  **Question: What is the % operator?**  **Answer:** It is referred to as the modulo or remainder operator. It returns the remainder of dividing the first operand by the second operand.  **Question: When can an object reference be cast to an interface reference?**  **Answer:** An object reference be cast to an interface reference when the object implements the referenced interface.  **Question: What is the difference between a Window and a Frame?**  **Answer:** The Frame class extends Window to define a main [application](http://www.roseindia.net/interviewquestions/corejava/difference-indow-frame.shtml) window that can have a menu bar.  **Question: Which class is extended by all other classes?**  **Answer:** The Object class is extended by all other classes.  **Question: Can an object be garbage collected while it is still reachable?**  **Answer:** A reachable object cannot be garbage collected. Only unreachable objects may be garbage collected..  **Question: Is the ternary operator written x : y ? z or x ? y : z ?**  **Answer:** It is written x ? y : z.  **Question: What is the difference between the Font and FontMetrics classes?**  **Answer:** The FontMetrics class is used to define implementation-specific properties, such as ascent and descent, of a [Font](http://www.roseindia.net/interviewquestions/corejava/difference-indow-frame.shtml) object.  **Question: How is rounding performed under integer division?**  **Answer:** The fractional part of the result is truncated. This is known as rounding toward zero.  **Question: What is the difference between the Reader/Writer class hierarchy and the InputStream/OutputStream class hierarchy?**  **Answer:** The Reader/Writer class hierarchy is character-oriented, and the InputStream/OutputStream class hierarchy is byte-oriented.  **Question: What classes of exceptions may be caught by a catch clause?**  **Answer:** A catch clause can catch any exception that may be assigned to the Throwable type. This includes the Error and Exception types.  **Question: If a class is declared without any access modifiers, where may the class be accessed?**  **Answer:** A class that is declared without any access modifiers is said to have package access. This means that the class can only be accessed by other classes and interfaces that are defined within the same package.  **Question: What is the SimpleTimeZone class?**  **Answer:** The SimpleTimeZone class provides [support](http://www.roseindia.net/interviewquestions/corejava/difference-indow-frame.shtml) for a Gregorian calendar.  **Question: What is the Map interface?**  **Answer:** The Map interface replaces the [JDK](http://www.roseindia.net/interviewquestions/corejava/difference-indow-frame.shtml) 1.1 Dictionary class and is used associate keys with values.  **Question: Does a class inherit the constructors of its superclass?**  **Answer:** A class does not inherit constructors from any of its superclasses.  **Question: For which statements does it make sense to use a label?**  **Answer:** The only statements for which it makes sense to use a label are those statements that can enclose a break or continue statement.  **Question: What is the purpose of the System class?**  **Answer:** The purpose of the [System](http://www.roseindia.net/interviewquestions/corejava/difference-indow-frame.shtml) class is to provide access to system resources.  **Question: Which TextComponent method is used to set a TextComponent to the read-only state?**  **Answer:** setEditable()  **Question: How are the elements of a CardLayout organized?**  **Answer:** The elements of a CardLayout are stacked, one on top of the other, like a deck of cards.  **Question: Is &&= a valid Java operator?**  **Answer:** No, it is not.  **Question: Name the eight primitive Java types.**  **Answer:** The eight primitive types are byte, char, short, int, long, float, double, and boolean.  **Question: Which class should you use to obtain design information about an object?**  **Answer:** The Class class is used to obtain information about an object's design.  **Question: What is the relationship between clipping and repainting?**  **Answer:** When a window is repainted by the AWT painting thread, it sets the clipping regions to the area of the window that requires repainting.  **Question: Is "abc" a primitive value?**  **Answer:** The String literal "abc" is not a primitive value. It is a String object.  **Question: What is the relationship between an event-listener interface and an event-adapter class?**  **Answer:** An event-listener interface defines the methods that must be implemented by an event handler for a particular kind of event. An event adapter provides a default implementation of an event-listener interface.  **Question: What restrictions are placed on the values of each case of a switch statement?**  **Answer:** During compilation, the values of each case of a switch statement must evaluate to a value that can be promoted to an int value.  **Question: What modifiers may be used with an interface declaration?**  **Answer:** An interface may be declared as public or abstract.  **Question: Is a class a subclass of itself?**  **Answer:** A class is a subclass of itself.  **Question: What is the highest-level event class of the event-delegation model?**  **Answer:** The java.util.EventObject class is the highest-level class in the event-delegation class hierarchy.  **Question: What event results from the clicking of a button?**  **Answer:** The ActionEvent event is generated as the result of the clicking of a button.  **Question: How can a GUI component handle its own events?**  **Answer:** A component can handle its own events by implementing the required event-listener interface and adding itself as its own event listener.  **Question: What is the difference between a while statement and a dostatement?**  **Answer:** A while statement checks at the beginning of a loop to see whether the next loop iteration should occur. A do statement checks at the end of a loop to see whether the next iteration of a loop should occur. The do statement will always execute the body of a loop at least once.  **Question: How are the elements of a GridBagLayout organized?**  **Answer:** The elements of a GridBagLayout are organized according to a grid. However, the elements are of different sizes and may occupy more than one row or column of the grid. In addition, the rows and columns may have different sizes.  **Question: What advantage do Java's layout managers provide over traditional windowing systems?**  **Answer:** Java uses layout managers to lay out components in a consistent manner across all windowing platforms. Since Java's layout managers aren't tied to absolute sizing and positioning, they are able to accomodate platform-specific differences among windowing systems.  **Question: What is the Collection interface?**  **Answer:** The Collection interface provides support for the implementation of a mathematical bag - an unordered collection of objects that may contain duplicates.  **Question: What modifiers can be used with a local inner class?**  **Answer:** A local inner class may be final or abstract.  **Question: What is the difference between static and non-static variables?**  **Answer:** A static variable is associated with the class as a whole rather than with specific instances of a class. Non-static variables take on unique values with each object instance.  **Question: What is the difference between the paint() and repaint() methods?**  **Answer:** The paint() method [supports](http://www.roseindia.net/interviewquestions/corejava/static-non-static-variables.shtml) painting via a Graphics object. The repaint() method is used to cause paint() to be invoked by the AWT painting thread.  **Question: What is the purpose of the File class?**  **Answer:** The File class is used to create objects that provide access to the files and directories of a local file [system](http://www.roseindia.net/interviewquestions/corejava/static-non-static-variables.shtml).  **Question: Can an exception be rethrown?**  **Answer:** Yes, an exception can be rethrown.  **Question: Which Math method is used to calculate the absolute value of a number?**  **Answer:** The abs() method is used to calculate absolute values.  **Question: When does the compiler supply a default constructor for a class?**  **Answer:** The [compiler](http://www.roseindia.net/interviewquestions/corejava/static-non-static-variables.shtml) supplies a default constructor for a class if no other constructors are provided.  **Question: When is the finally clause of a try-catch-finally statement executed?**  **Answer:** The finally clause of the try-catch-finally statement is always executed unless the thread of execution terminates or an exception occurs within the execution of the finally clause.  **Question: Which class is the immediate superclass of the Container class?**  **Answer:** Component  **Question: If a method is declared as protected, where may the method be accessed?**  **Answer:** A protected method may only be accessed by classes or [interfaces](http://www.roseindia.net/interviewquestions/corejava/static-non-static-variables.shtml) of the same package or by subclasses of the class in which it is declared.  **Question: How can the Checkbox class be used to create a radio button?**  **Answer:** By associating Checkbox objects with a CheckboxGroup.  **Question: Which non-Unicode letter characters may be used as the first character of an identifier?**  **Answer:** The non-Unicode letter characters $ and \_ may appear as the first character of an identifier  **Question: What restrictions are placed on method overloading?**  **Answer:** Two methods may not have the same name and argument list but different return types.  **Question: What is casting?**  **Answer:** There are two types of casting, casting between primitive numeric types and casting between object references. Casting between numeric types is used to convert larger values, such as double values, to smaller values, such as byte values. Casting between object references is used to refer to an object by a compatible class, interface, or array type reference.  **Question: What is the return type of a program's main() method?**  **Answer:** A program's main() method has a void return type.  **Question: Name four Container classes.**  **Answer:** Window, Frame, Dialog, File Dialog, Panel, [Applet](http://www.roseindia.net/interviewquestions/corejava/container-classes.shtml), or ScrollPane  **Question: What is the difference between a Choice and a List?**  **Answer:** A Choice is displayed in a compact form that requires you to pull it down to see the list of available choices. Only one item may be selected from a Choice. A List may be displayed in such a way that several [List items](http://www.roseindia.net/interviewquestions/corejava/container-classes.shtml) are visible. A List supports the selection of one or more List items.  **Question: What class of exceptions are generated by the Java run-time system?**  **Answer:** The Java runtime [system](http://www.roseindia.net/interviewquestions/corejava/container-classes.shtml) generates Runtime Exception and Error exceptions.  **Question: What class allows you to read objects directly from a stream?**  **Answer:** The ObjectInputStream class supports the reading of objects from input streams.  **Question: What is the difference between a field variable and a local variable?**  **Answer:** A field variable is a variable that is declared as a member of a class. A local variable is a variable that is declared local to a method.  **Question: Under what conditions is an object's finalize() method invoked by the garbage collector?**  **Answer:** The garbage collector invokes an object's finalize() method when it detects that the object has become unreachable.  **Question: How are this() and super() used with constructors?**  **Answer:** this() is used to invoke a constructor of the same class. super() is used to invoke a superclass constructor.  **Question: What is the relationship between a method's throws clause and the exceptions that can be thrown during the method's execution?**  **Answer:** A method's throws clause must declare any checked exceptions that are not caught within the body of the method.  **Question: What is the difference between the JDK 1.02 event model and the event-delegation model introduced with JDK 1.1?**  **Answer:** The JDK 1.02 [event model](http://www.roseindia.net/interviewquestions/corejava/container-classes.shtml) uses an event inheritance or bubbling approach. In this model, components are required to handle their own events. If they do not handle a particular event, the event is inherited by (or bubbled up to) the component's container. The container then either handles the event or it is bubbled up to its container and so on, until the highest-level container has been tried.  In the event-delegation model, specific objects are designated as event handlers for GUI components. These objects implement event-listener interfaces. The event-delegation model is more efficient than the event-inheritance model because it eliminates the processing required to support the bubbling of unhand led events.  **Question: How is it possible for two String objects with identical values not to be equal under the == operator?**  **Answer:** The == operator compares two objects to determine if they are the same object in memory. It is possible for two String objects to have the same value, but located indifferent areas of memory.  **Question: Why are the methods of the Math class static?**  **Answer:** So they can be invoked as if they are a mathematical code library.  **Question: What Checkbox method allows you to tell if a Checkbox is checked?**  **Answer:** getState()  **Question: What are the legal operands of the instance of operator?**  **Answer:** The left operand is an object reference or null value and the right operand is a class, interface, or array type.  **Question: How are the elements of a GridLayout organized?**  **Answer:** The elements of a GridBad layout are of equal size and are laid out using the squares of a grid.  **Question: What an I/O filter?**  **Answer:** An I/O filter is an object that reads from one stream and writes to another, usually altering the data in some way as it is passed from one stream to another.  **Question: If an object is garbage collected, can it become reachable again?**  **Answer:** Once an object is garbage collected, it ceases to exist.It can no longer become reachable again.  **Question: What is the Set** [**interface**](http://www.roseindia.net/interviewquestions/corejava/grid-Layout-organized.shtml)**?**  **Answer:** The Set interface provides methods for accessing the elements of a finite mathematical set. Sets do not allow duplicate elements.  **Question: What classes of exceptions may be thrown by a throw statement?**  **Answer:** A throw statement may throw any expression that may be assigned to the Throwable type.  **Question: What are E and PI?**  **Answer:** E is the base of the natural logarithm and PI is mathematical value pi.  **Question: Are true and false keywords?**  **Answer:** The values true and false are not [keywords](http://www.roseindia.net/interviewquestions/corejava/grid-Layout-organized.shtml).  **Question: What is a void return type?**  **Answer:** A void return type indicates that a method does not return a value.  **Question: What is the purpose of the enableEvents() method?**  **Answer:** The enableEvents() method is used to enable an event for a particular object. Normally, an event is enabled when a listener is added to an object for a particular event. The enableEvents() method is used by objects that handle events by overriding their event-dispatch methods.  **Question: What is the difference between the File and RandomAccessFile classes?**  **Answer:** The File class encapsulates the files and directories of the local file [system](http://www.roseindia.net/interviewquestions/corejava/grid-Layout-organized.shtml). The RandomAccessFile class provides the methods needed to directly [access data](http://www.roseindia.net/interviewquestions/corejava/grid-Layout-organized.shtml) contained in any part of a file.  **Question: What happens when you add a double value to a String?**  **Answer:** The result is a String object.  **Question: What is your platform's default character encoding?**  **Answer:** If you are running [Java](http://www.roseindia.net/interviewquestions/corejava/grid-Layout-organized.shtml) on English Windows platforms, it is probably Cp1252. If you are running Java on English Solaris platforms, it is most likely 8859\_1..  **Question: Which package is always imported by default?**  **Answer:** The java.lang package is always imported by default.  **Question: What interface must an object implement before it can be written to a stream as an object?**  **Answer:** An object must implement the Serializable or Externalizable interface before it can be written to a stream as an object.  **Question: How are this and super used?**  **Answer:** this is used to refer to the current object instance. super is used to refer to the variables and methods of the superclass of the current object instance.  **Question: What is the purpose of garbage collection?**  **Answer:** The purpose of garbage collection is to identify and discard objects that are no longer needed by a program so that their resources may be reclaimed and reused.  **Question: What is a compilation unit?**  **Answer:** A compilation unit is a [Java source code](http://www.roseindia.net/interviewquestions/corejava/compilation-unit.shtml) file.  **Question: What interface is extended by AWT event listeners?**  **Answer:** All AWT event listeners extend the [java](http://www.roseindia.net/interviewquestions/corejava/compilation-unit.shtml).util.EventListener interface.  **Question: What restrictions are placed on method overriding?**  **Answer:** Overridden methods must have the same name, argument list, and return type. The overriding method may not limit the access of the method it overrides. The overriding method may not throw any exceptions that may not be thrown by the overridden method.  **Question: What happens if an exception is not caught?**  **Answer:** An uncaught exception results in the uncaughtException() method of the thread's ThreadGroup being invoked, which eventually results in the termination of the program in which it is thrown.  **Question: What is a layout manager?**  **Answer:** A layout manager is an object that is used to organize components in a container.  **Question: Which arithmetic operations can result in the throwing of an ArithmeticException?**  **Answer:** Integer / and % can result in the throwing of an ArithmeticException.  **Question: Can an abstract class be final?**  **Answer:** An abstract class may not be declared as final.  **Question: What is the ResourceBundle class?**  **Answer:** The ResourceBundle class is used to store locale-specific resources that can be loaded by a program to tailor the program's appearance to the particular locale in which it is being run.  **Question: What happens if a try-catch-finally statement does not have a catch clause to handle an exception that is thrown within the body of the try statement?**  **Answer:** The exception propagates up to the next higher level try-catch statement (if any) or results in the program's termination.  **Question: What is numeric promotion?**  **Answer:** Numeric promotion is the conversion of a smaller numeric type to a larger numeric type, so that integer and floating-point [operations](http://www.roseindia.net/interviewquestions/corejava/compilation-unit.shtml) may take place. In numerical promotion, byte, char, and short values are converted to int values. The int values are also converted to long values, if necessary. The long and float values are converted to double values, as required.  **Question: What is the difference between a Scrollbar and a ScrollPane?**  **Answer:** A [Scrollbar](http://www.roseindia.net/interviewquestions/corejava/compilation-unit.shtml) is a Component, but not a Container. A ScrollPane is a Container. A ScrollPane handles its own events and performs its own scrolling.  **Question: What is the difference between a public and a non-public class?**  **Answer:** A public class may be accessed outside of its package. A non-public class may not be accessed outside of its package.  **Question: To what value is a variable of the boolean type automatically initialized?**  The default value of the boolean type is false.  **Question: Can try statements be nested?**  Try statements may be tested.  **Question: What is the difference between the prefix and postfix forms of the ++ operator?**  The prefix form performs the increment operation and returns the value ofthe increment operation. The postfix form returns the current value all of the expression and then performs the increment operation on that value.  **Question: What is the purpose of a statement block?**  A statement block is used to organize a sequence of statements as a single statement group.  **Question: What is a** [**Java**](http://www.roseindia.net/interviewquestions/corejava/automatically-initialized.shtml) **package and how is it used?**  A Java package is a naming context for classes and interfaces. A package is used to create a separate name space for groups of classes and interfaces. Packages are also used to organize related classes and interfaces into a single API unit and to control accessibility to these classes and interfaces.  **Question: What modifiers may be used with a top-level class?**  A top-level class may be public, abstract, or final.  **Question: What are the Object and Class classes used for?**  The Object class is the highest-level class in the [Java class](http://www.roseindia.net/interviewquestions/corejava/automatically-initialized.shtml) hierarchy. The Class class is used to represent the classes and interfaces that are loaded by a [Java program](http://www.roseindia.net/interviewquestions/corejava/automatically-initialized.shtml).  **Question: How does a try statement determine which catch clause should be used to handle an exception?**  When an exception is thrown within the body of a try statement, the catch clauses of the try statement are examined in the order in which they appear. The first catch clause that is capable of handling the exception is executed. The remaining catch clauses are ignored.  **Question: Can an unreachable object become reachable again?**  **Answer:** An unreachable object may become reachable again. This can happen when the object's finalize() method is invoked and the object performs an operation which causes it to become accessible to reachable objects.  **Question: When is an object subject to garbage collection?**  **Answer:** An object is subject to garbage collection when it becomes unreachable to the program in which it is used.  **Question: What methods are used to get and set the text label displayed by a Button object?**  **Answer:** getLabel() and setLabel()  **Question: Which Component subclass is used for drawing and painting?**  **Answer:** Canvas  **Question: What are the problems faced by Java programmers who don't use layout managers?**  **Answer:** Without layout managers, [Java programmers](http://www.roseindia.net/interviewquestions/corejava/automatically-initialized.shtml) are faced with determining how their GUI will be displayed across multiple windowing [systems](http://www.roseindia.net/interviewquestions/corejava/automatically-initialized.shtml) and finding a common sizingand positioning that will work within the constraints imposed by each windowing system.  **Question: What is the difference between an if statement and a switch statement?**  **Answer:** The if statement is used to select among two alternatives. It uses a boolean expression to decide which alternative should be executed. The switch statement is used to select among multiple alternatives. It uses an int expression to determine which alternative should be executed.  **Question: What happens when you add a double value to a String?**  **Answer:** The result is a String object.  **Question: What is the List interface?**  **Answer:** The List interface provides support for ordered collections of objects.  **Question: Why do we need public static void main(String args[]) method in Java**  **Answer:** We need   * public: The method can be accessed outside the class / package * static: You need not have an instance of the class to access the method * void: Your application need not return a value, as the JVM launcher would return the value when it exits * main(): This is the entry point for the application   If the main() was not static, you would require an instance of the class in order to execute the method.  If this is the case, what would create the instance of the class? What if your class did not have a public constructor?   **Question: What is the difference between an Interface and an Abstract class**  **Answer:** In abstract class you can define as well as declare methods, the methods which are declared are to be marked as abstract.  In interface all we just declare methods and the definition is provided by the class which is implementing it   **Question: Explain serialization**  **Answer:** [Serialization](http://www.roseindia.net/interviewquestions/corejava/method-in-java.shtml) means storing a state of a [java object](http://www.roseindia.net/interviewquestions/corejava/method-in-java.shtml) by coverting it to byte stream   **Question: What are the rules of serialization**  **Answer:** **Rules:**  1. Static fileds are not serialized because they are not part of any one particular object  2. Fileds from the base class are handled only if hose are serializable  3. Transient fileds are not serialized   **Question: What is difference between error and exception**  **Answer:** Error occurs at runtime and cannot be recovered, Outofmemory is one such example. Exceptions on the other hand are due conditions which the application encounters such as FileNotFound exception or IO exceptions   **Question: What do you mean by object oreiented programming**  **Answer:** In object oreinted programming the emphasis is more on data than on the procedure and the program is divided into objects.  The data fields are hidden and they cant be accessed by external functions.  The design approach is bottom up.  The functions operate on data that is tied together in data structure   **Question: What are 4 pillars of object oreinted programming**  **Answer:**  1. Abstraction  It means hiding the details and only exposing the essentioal parts   2. Polymorphism  Polymorphism means having many forms. In [java](http://www.roseindia.net/interviewquestions/corejava/method-in-java.shtml) you can see polymorphism when you have multiple methods with the same name   3. Inheritance  Inheritance means the child class inherits the non private properties of the parent class   4. Encapsulation  It means data hiding. In java with encapsulate the data by making it private and even we want some other class to work on that data then the setter and getter methods are provided  **Question: Difference between procedural and object oreinted language**  **Answer:** In procedural programming the instructions are executed one after another and the data is exposed to the whole program  In OOPs programming the unit of program is an object which is nothing but combination of data and code and the data is not exposed outside the object   **Question: What is the difference between constructor and method**  **Answer:** Constructor will be automatically invoked when an object is created whereas method has to be called explicitly.   **Question: What is the difference between parameters and arguments**  **Answer:** While defining method, variables passed in the method are called parameters. While using those methods, values passed to those variables are called arguments.   **Question: What is reflection in java**  **Answer:** Reflection allows Java code to discover information about the fields, methods and constructors of loaded classes and to dynamically invoke them   **Question: What is a cloneable interface and how many methods does it contain**  **Answer:** It is not having any method because it is a TAGGED or MARKER interface   **Question: What's the difference between a queue and a stack**  **Answer:** Stacks works by last-in-first-out rule (LIFO), while queues use the FIFO rule   **Question: Can you make an instance of abstract class**  **Answer:** No you cannot create an instance of abstract class   **Question: What are parsers**  **Answer:** Parsers are used for processing XML documents. There are 2 types of parsers DOM parser and SAX Parser   **Question: Difference between SAX and DOM parser**  **Answer:** DOM parsers are Object based and SAX parsers are event based  DOM parsers creates Tree in the memory whereas SAX parser does not and hence it is faster than DOM  DOM parser are useful when we have to modify the XML, with SAX parser you cannot modify the xml, it is read only   **Question: What is the difference between Java Bean and Java Class**  **Answer:** Basically a Bean is a [java class](http://www.roseindia.net/interviewquestions/corejava/method-in-java.shtml) but it has getter and setter method and it does not have any logic in it, it is used for holding data.  On the other hand the Java class can have what a [java bean](http://www.roseindia.net/interviewquestions/corejava/method-in-java.shtml) has and also has some logic inside it  **Question: What are null or Marker interfaces in** [**Java**](http://www.roseindia.net/interviewquestions/corejava/null-marker-interfaces-in-java.shtml)  **Answer:** The null interfaces are marker interfaces, they do not have function declarations in them, they are empty interfaces, this is to convey the compiler that they have to be treated differently   **Question: Does java Support multiple inheritance**  **Answer:** Java does not support multiple inheritance directly like C++, because then it is prone to ambiguity, example if a class extends 2 other classes and these 2 parent classes have same method names then there is ambiguity. Hence in Java Multiple inheritance is supported using Interfaces   **Question: What are virtual function**  **Answer:** In OOP when a [derived class](http://www.roseindia.net/interviewquestions/corejava/null-marker-interfaces-in-java.shtml) inherits from a base class, an object of the derived class may be referred to (or cast) as either being the base class type or the derived class type. If there are base class functions overridden by the derived class, a problem then arises when a derived object has been cast as the base class type. When a derived object is referred to as being of the base's type, the desired function call behavior is ambiguous.   The distinction between virtual and not virtual is provided to solve this issue. If the function in question is designated "virtual" then the derived class's function would be called (if it exists). If it is not virtual, the base class's function would be called.   **Question: Does java support virtual functions**  **Answer:** No java does not support virtual functions direclty like in C++, but it supports using Abstract class and interfaces   **Question: Describe what happens when an object is created in Java**  **Answer:** Several things happen in a particular order to ensure the object is constructed properly:   1. Memory is allocated from heap to hold all instance variables and implementation-specific data of the  object and its superclasses. Implemenation-specific data includes pointers to class and method data.   2. The instance variables of the objects are initialized to their default values.   3. The constructor for the most derived class is invoked. The first thing a constructor does is call the  consctructor for its superclasses. This process continues until the constrcutor for java.lang.Object is called,  as java.lang.Object is the base class for all objects in java.   4. Before the body of the constructor is executed, all instance variable initializers and initialization blocks  are executed. Then the body of the constructor is executed. Thus, the constructor for the base class  completes first and constructor for the most derived class completes last.  **Question: What is the purpose of System Class**  **Answer:** The purpose of the system class is to provide the access to the System reources   **Question: What is *instanceOf* operator used for**  **Answer:** It is used to if an object can be cast into a specific type without throwing Class cast exception   **Question: Why we should not have instance variable in an interface**  **Answer:** Since all data fields and methods in an Interface are public by default, then we implement that interface in our class then we have public members in our class and this class will expose these data members and this is violation of encapsulation as now the data is not secure   **Question: What is JVM**  **Answer:** When we install a java package. It contains 2 things  \* The Java Runtime Environment (JRE)  \* The [Java Development](http://www.roseindia.net/interviewquestions/corejava/null-marker-interfaces-in-java.shtml) Kit (JDK)   The JRE provides runtime support for Java applications. The JDK provides the Java compiler and other development tools. The JDK includes the JRE.   Both the JRE and the JDK include a Java Virtual Machine (JVM). This is the application that executes a [Java program](http://www.roseindia.net/interviewquestions/corejava/null-marker-interfaces-in-java.shtml). A Java program requires a JVM to run on a particular platform   **Question: Can abstract class be final**  **Answer:** No, abstract class cannot be final   **Question: When a new object of derived Class is created, whose constructor will be called first, childs or parents**  **Answer:** Even when the new object of child class is created, first the Base class constructor gets executed and then the child classes constructor   **Question: What is a singleton class**  **Answer:** A singleton is an object that cannot be instantiated. The restriction on the singleton is that there can be only one instance of a singleton created by the Java Virtual Machine (JVM) - by prevent direct instantiation we can ensure that developers don't create a second copy.   **Question: Can an abstract class have final method**  **Answer:** Yes   **Question: Can a final class have an abstract method**  **Answer:** No, the compiler will give an error   **Question: What is the difference between Authentication and Authorization**  **Answer:** Authentication is a process for verifying that an individual is who they say they are. Authorization is an additional level of [security](http://www.roseindia.net/interviewquestions/corejava/null-marker-interfaces-in-java.shtml), and it means that a particular user (usually authenticated), may have access to a particular resource say record, file, directory or script. **Objects and Classes**  **Question: What's the difference between constructors and other methods**  **Answer:** Constructors must have the same name as the class and can not return a value. They are only called once while regular methods could be called many times.   **Question: What is the difference between Overloading and Overriding**  **Answer:** Overloading : Reusing the same method name with different arguments and perhaps a different return type is called as overloading  Overriding : Using the same method name with identical arguments and return type is know as overriding   **Question: What do you understand by late binding or virtual method Invocation. (Example of runtime polymorphism)**  **Answer:** When a [compiler](http://www.roseindia.net/interviewquestions/corejava/objects-classes.shtml) for a non object oriented [language](http://www.roseindia.net/interviewquestions/corejava/objects-classes.shtml) comes across a method invocation, it determines exactly what target code should be called and build machine language to represent that call. In an object oriented language, this is not possible since the proper code to invoke is determined based upon the class if the object being used to make the call, not the type of the variable. Instead code is generated that will allow the decision to be made at run time. This delayed decision making is called as late binding   **Question: Can overriding methods have different return types**  **Answer:** No they cannot have different return types   **Question: If the method to be overridden has access type protected, can subclass have the access type as private**  **Answer:** No, it must have access type as protected or public, since an overriding method must not be less accessible than the method it overrides   **Question: Can constructors be overloaded**  **Answer:** Yes constructors can be overloaded   **Question: What happens when a constructor of the subclass is called**  **Answer:** A constructor delays running its body until the parent parts of the class have been initialized. This commonly happens because of an implicit call to super() added by the compiler. You can provide your own call to super(arguments..) to control the way the parent parts are initialized. If you do this, it must be the first statement of the constructor.   **Question: If you use super() or this() in a constructor where should it appear in the constructor**  **Answer:** It should always be the first statement in the constructor   **Question: What is an inner class**  **Answer:** An inner class is same as any other class, but is declared inside some other class   **Question: How will you reference the inner class**  **Answer:** To reference it you will have to use OuterClass$InnerClass   **Question: Can objects that are instances of inner class access the members of the outer class**  **Answer:** Yes they can access the members of the outer class   **Question: What modifiers may be used with an inner class that is a member of an outer class?**  **Answer:** A (non-local) inner class may be declared as public, protected, private, static, final, or abstract   **Question: Can inner classes be static**  **Answer:** Yes inner [classes](http://www.roseindia.net/interviewquestions/corejava/objects-classes.shtml) can be static, but they cannot access the non static data of the outer classes, though they can access the static data   **Question: Can an inner class be defined inside a method**  **Answer:** Yes it can be defined inside a method and it can [access data](http://www.roseindia.net/interviewquestions/corejava/objects-classes.shtml) of the enclosing methods or a formal parameter if it is final   **Question: What is an anonymous class**  **Answer:** Some classes defined inside a method do not need a name, such classes are called anonymous classes   **Question: What are the rules of anonymous class**  **Answer:** The class is instantiated and declared in the same place The declaration and instantiation takes the form new Xxxx () {// body}  Where Xxxx is an interface name. An anonymous class cannot have a constructor. Since you do not specify a name for the class, you cannot use that name to specify a constructor  **Question: What is the difference between “= =” and “equals()”**  **Answer:** “= =” does shallow comparison, It retuns true if the two object points to the same address in the memory, i.e if the same the same reference  “equals()” does deep comparison, it checks if the values of the data in the object are same   **Question: What would you use to compare two String variables - the operator == or the method equals()?**  **Answer:** I'd use the method equals() to compare the values of the Strings and the == to check if two variables point at the same instance of a String object   **Question: Give example of a final class**  **Answer:** Math class is final class and hence cannot be extended   **Question: What is the difference between String and StringBuffer**  **Answer:** String is an immutable class, i.e you cannot change the values of that class  **Example:**  String str = “[java](http://www.roseindia.net/interviewquestions/corejava/java-util-packages.shtml)”; // address in memory say 12345  And now if you assign a new value to the variable str then  str = “core java”; then the value of the variable at address 12345 will not change but a new memory is allocated for this variable say 54321  So in the memory address 12345 will have value “java”  And the memory address 54321 will have value “core java” and the variable str will now be pointing to address 54321 in memory   StringBuffer can be modified dynamically  Example:  StringBuffer strt =”java” // address in memory is say 12345  And now if you assign a new value to the variable str then  Str = “core java”; then value in the address of memory will get replaced, a new memory address is not allocated in this case.   **Question: What will be the result if you compare StringBuffer with String if both have same values**  **Answer:** It will return false as you cannot compare String with StringBuffer |

Q.How are this() and super() used with constructors?

This() is used to invoke a constructor of the same class. super() is used to invoke a superclass constructor

Q.What is the difference between static and non-static variables?

### **A static variable is associated with the class as a whole rather than with specific instances of a class. Non-static variables take on unique values with each object instance. For Example : public class Test{ static int i=5; int j=7; } Test t1= new Test (); Test t2= new Test (); t1.j=10; t2.j=15; but for the case of static i , t1.i=10; t2.i=15; System.out.println(t2.i) will give you 15 not 10 . because i is sharable and related to class not instance. Q.What is the purpose of finalization?**

The purpose of finalization is to give an unreachable object the opportunity to perform any cleanup processing before the object is garbage collected.

Q.Does garbage collection guarantee that a program will not run out of memory?

Garbage collection does not guarantee that a program will not run out of memory. It is possible for programs to use up memory resources faster than they are garbage collected. It is also possible for programs to create objects that are not subject to garbage collection

Q.If I write System.exit (0); at the end of the try block, will the finally block still execute?

No ,in this case the finally block will not execute because when you say System.exit (0); the control immediately goes out of the program, and thus finally never executes.

Q: What is wrapper class? Explain with example?

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.   
All wrapper classes are final. You can't extend the class.

Q. What are different types of inner classes?

Nested top-level classes, Member classes, Local classes, Anonymous classes   
  
Nested top-level classes- If you declare a class within a class and specify the static modifier, the compiler treats the class just like any other top-level class.   
Any class outside the declaring class accesses the nested class with the declaring class name acting similarly to a package. eg, outer.inner. Top-level inner classes implicitly have access only to static variables.There can also be inner interfaces. All of these are of the nested top-level variety.   
  
Member classes - Member inner classes are just like other member methods and member variables and access to the member class is restricted, just like methods and variables. This means a public member class acts similarly to a nested top-level class. The primary difference between member classes and nested top-level classes is that member classes have access to the specific instance of the enclosing class.   
  
Local classes - Local classes are like local variables, specific to a block of code. Their visibility is only within the block of their declaration. In order for the class to be useful beyond the declaration block, it would need to implement a   
more publicly available interface.Because local classes are not members, the modifiers public, protected, private, and static are not usable.   
  
Anonymous classes - Anonymous inner classes extend local inner classes one level further. As anonymous classes have no name, you cannot provide a constructor.

Q. What is Overriding?

When a class defines a method using the same name, return type, and arguments as a method in its superclass, the method in the class overrides the method in the superclass.   
When the method is invoked for an object of the class, it is the new definition of the method that is called, and not the method definition from superclass.   
Rules to folow :   
Methods may be overridden to be more public, not more private.   
Methods may be overridden to be thows the same exception or subclass of the exception thrown by the method of superclass.

Q. What is final?

A final class can't be extended ie., final class may not be subclassed. A final method can't be overridden when its class is inherited. You can't change value of a final variable (is a constant).

.What is static in java?

Static means one per class, not one for each object no matter how many instance of a class might exist. This means that you can use them without creating an instance of a class.   
Static methods are implicitly final, because overriding is done based on the type of the object, and static methods are attached to a class, not an object.   
A static method in a superclass can be shadowed by another static method in a subclass, as long as the original method was not declared final.   
However, you can't override a static method with a nonstatic method.   
In other words, you can't change a static method into an instance method in a subclass.   
Static varibales are not serialized.

What is an abstract class?

Abstract class must be extended/subclassed (to be useful). It serves as a template. A class that is abstract may not be instantiated (ie, you may not call its constructor), abstract class may contain static data. Any class with an abstract method is automatically abstract itself, and must be declared as such.   
A class may be declared abstract even if it has no abstract methods. This prevents it from being instantiated

Q. What are the modifiers in Java ?

public : Public class is visible in other packages, field is visible everywhere (class must be public too)   
private : Private variables or methods may be used only by an instance of the same class that declares the variable or method, A private feature may only be accessed by the class that owns the feature.   
protected : Is available to all classes in the same package and also available to all subclasses of the class that owns the protected feature.This access is provided even to subclasses that reside in a different package from the class that owns the protected feature.   
default :What you get by default ie, without any access modifier (ie, public private or protected).It means that it is visible to all within a particular package

.

Q.What are pass by reference and passby value in Java?

Pass By Reference means the passing the address itself rather than passing the value.   
Passby Value means passing a copy of the value to be passed.   
Java only supports pass by value. With objects, the object reference itself is passed by value and so both the original reference and parameter copy both refer to the same object.

Q. When we go for Abstract and Interface in Java?

When the sub-types behaviour is totally different then you use an interface, when the sub-types behaviour is partially common and different with respect to the supertype an abstract class is used.   
In an abstract class the partially common behaviour is given a concrete implementation.   
Since there is no common behaviour between an interface and a sub-type an interface does not have an implementation for any of its behaviour.   
For Example - Abstract Class   
abstract public class Animal {   
public eatChicken(){   
System.out.println("Eat Chicken");   
}   
  
}   
  
public Lion extends Animal{   
}   
public Tiger extends Animal{   
}   
  
Both Lion and Tiger class has common behaviour eatChicken() and all the implementaion put into eatChicken() method of super calss.   
so we go for abstract class.   
  
Example- Interface   
  
Interface Animal{   
public eat();   
}   
public Lion implements Animal{   
public eat(){   
System.out.println("Lion eat Non Veg");   
}   
}   
public Cow implements Animal{   
public eat(){   
System.out.println("Cow eat Veg");   
}   
}   
Both Lion and Cow class has different behaviour for eat() and they have different implementations. so we go for Interface.

Q.What is the difference between interface and abstract class?

interface contains methods that must be abstract; abstract class may contain concrete methods.   
\* interface contains variables that must be static and final; abstract class may contain non-final and final variables.   
\* members in an interface are public by default, abstract class may contain non-public members.   
\* interface is used to "implements"; whereas abstract class is used to "extends".   
\* interface can be used to achieve multiple inheritance; abstract class can be used as a single inheritance.   
\* interface can "extends" another interface, abstract class can "extends" another class and "implements" multiple interfaces.   
\* interface is absolutely abstract; abstract class can be invoked if a main() exists.   
\* interface is more flexible than abstract class because one class can only "extends" one super class, but "implements" multiple interfaces.   
\* If given a choice, use interface instead of abstract class.

Q.What are the Garbage collection algorithms in Java?

Any garbage collection algorithm must do two basic things.   
First, it must detect garbage objects.   
Second, it must reclaim the heap space used by the garbage objects and make it available to the program.   
Garbage detection is ordinarily accomplished by defining a set of roots and determining reachability from the roots.   
An object is reachable if there is some path of references from the roots by which the executing program can access the object.   
The roots are always accessible to the program. Any objects that are reachable from the roots are considered live.   
Objects that are not reachable are considered garbage, because they can no longer affect the future course of program execution.   
  
    a. Reference counting collectors   
Reference counting was an early garbage collection strategy.   
here a reference count is maintained for each object.   
When an object is first created its reference count is set to one.   
When any other object or root is assigned a reference to that object, the object's count is incremented.   
When a reference to an object goes out of scope or is assigned a new value, the object's count is decremented.   
Any object with a reference count of zero can be garbage collected.   
When an object is garbage collected, any objects that it refers to has their reference counts decremented.   
In this way the garbage collection of one object may lead to the subsequent garbage collection of other objects.   
  
b. Tracing collectors   
Tracing garbage collectors trace out the graph of object references starting with the root nodes.   
Objects that are encountered during the trace are marked in some way.   
Marking is generally done by either setting flags in the objects themselves or by setting flags in a separate bitmap.   
After the trace is complete, unmarked objects are known to be unreachable and can be garbage collected.   
The basic tracing algorithm is called mark and sweep.   
This name refers to the two phases of the garbage collection process.   
In the mark phase, the garbage collector traverses the tree of references and marks each object it encounters.   
In the sweep phase unmarked objects are freed, and the resulting memory is made available to the executing program.   
In the JVM the sweep phase must include finalization of objects.   
  
    c. Compacting collectors   
Garbage collectors of JVMs will likely have a strategy to combat heap fragmentation.   
Two strategies commonly used by mark and sweep collectors are compacting and copying.   
Both of these approaches move objects on the fly to reduce heap fragmentation.   
Compacting collectors slide live objects over free memory space toward one end of the heap.   
In the process the other end of the heap becomes one large contiguous free area.   
All references to the moved objects are updated to refer to the new location.   
  
    d. Copying collectors   
Copying garbage collectors move all live objects to a new area.   
As the objects are moved to the new area, they are placed side by side,   
thus eliminating any free spaces that may have separated them in the old area.   
The old area is then known to be all free space.   
The advantage of this approach is that objects can be copied as they are discovered by the traversal from the root nodes.   
There are no separate mark and sweep phases.   
Objects are copied to the new area on the fly, and forwarding pointers are left in their old locations.   
The forwarding pointers allow objects encountered later in the traversal that refer to already copied objects to know the new location of the copied objects.

Q.What is garbage collection and the purpose of garbage collection in Java?

The JVM's heap stores all objects created by an executing Java program.   
Objects are created by Java's "new" operator, and memory for new objects is allocated on the heap at run time.   
Garbage collection is the process of automatically freeing objects that are no longer referenced by the program.   
This frees the programmer from having to keep track of when to free allocated memory,   
thereby preventing many potential bugs and headaches.   
The name "garbage collection" implies that objects that are no longer needed by the program are "garbage" and can be thrown away.   
Other word we can say memory recycling.   
When an object is no longer referenced by the program, the heap space it occupies must be recycled so that the space is available for subsequent new objects.   
The garbage collector must somehow determine which objects are no longer referenced by the program and make available the heap space occupied by such unreferenced objects.   
In the process of freeing unreferenced objects, the garbage collector must run any finalizers of objects being freed.

### **Q. What is the difference between an Interface and an Abstract class?**

An abstract class can have instance methods that implement a default behavior. An Interface can only declare constants and instance methods, but cannot implement default behavior and all methods are implicitly abstract. An interface has all public members and no implementation. An abstract class is a class which may have the usual flavors of class members (private, protected, etc.), but has some abstract methods.

**Question:** What is transient variable?  
**Answer:** Transient variable can't be serialize. For example if a variable is declared as transient in a Serializable class and the class is written to an ObjectStream, the value of the variable can't be written to the stream instead when the class is retrieved from the ObjectStream the value of the variable becomes **null**.

**Question:** Is Iterator a Class or Interface? What is its use?  
**Answer:** Iterator is an interface which is used to step through the elements of a Collection. 

**Question:** What is similarities/difference between an Abstract class and Interface?  
**Answer:**  Differences are as follows:

* Interfaces provide a form of multiple inheritance. A class can extend only one other class.
* Interfaces are limited to public methods and constants with no implementation. Abstract classes can have a partial implementation, protected parts, static methods, etc.
* A Class may implement several interfaces. But in case of abstract class, a class may extend only one abstract class.
* Interfaces are slow as it requires extra indirection to to find corresponding method in in the actual class. Abstract classes are fast.

Similarities:

* Neither Abstract classes or Interface can be instantiated.

**Question:** How to define an Abstract class?  
**Answer:** A class containing abstract method is called Abstract class. An Abstract class can't be instantiated.   
Example of Abstract class:  
abstract class testAbstractClass {   
    protected String myString;   
    public String getMyString() {   
        return myString;   
        }   
    public abstract string anyAbstractFunction();  
}

**Question:** How to define an Interface?  
**Answer:** In [Java](http://www.roseindia.net/interviewquestions/corejava.shtml) Interface defines the methods but does not implement them. Interface can include constants. A class that implements the interfaces is bound to implement all the methods defined in Interface.  
Emaple of Interface:  
  
public interface sampleInterface {  
    public void functionOne();  
  
    public long CONSTANT\_ONE = 1000;   
}

**4.What are the advantages of using** [**exception handling**](http://www.developersbook.com/corejava/interview-questions/exception-interview-questions-faqs.php)**?**

Exception handling provides the following advantages over "traditional" error management techniques:

* Separating Error Handling Code from "Regular" Code.
* Propagating Errors Up the Call Stack.
* Grouping Error Types and Error Differentiation.

**5.What are the types of Exceptions in** [**Java**](http://www.developersbook.com/corejava/interview-questions/exception-interview-questions-faqs.php)

There are two types of exceptions in Java, unchecked exceptions and checked exceptions.

* **Checked exceptions**: A checked exception is some subclass of Exception (or Exception itself), excluding class RuntimeException and its subclasses. Each method must either handle all checked exceptions by supplying a catch clause or list each unhandled checked exception as a thrown exception.
* **Unchecked exceptions:** All Exceptions that extend the RuntimeException class are unchecked exceptions. Class Error and its subclasses also are unchecked.

**6.Why Errors are Not Checked?**

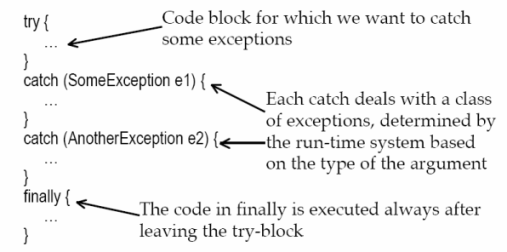
A unchecked exception classes which are the *error* classes (Error and its subclasses) are exempted from compile-time checking because they can occur at many points in the program and recovery from them is difficult or impossible. A program declaring such exceptions would be pointlessly.

**7.Why Runtime Exceptions are Not Checked?**

The *runtime exception* classes (RuntimeException and its subclasses) are exempted from compile-time checking because, in the judgment of the designers of the [Java programming language](http://www.developersbook.com/corejava/interview-questions/exception-interview-questions-faqs.php), having to declare such exceptions would not aid significantly in establishing the correctness of programs. Many of the operations and constructs of the Java programming language can result in runtime exceptions. The information available to a [compiler](http://www.developersbook.com/corejava/interview-questions/exception-interview-questions-faqs.php), and the level of analysis the compiler performs, are usually not sufficient to establish that such run-time exceptions cannot occur, even though this may be obvious to the programmer. Requiring such exception classes to be declared would simply be an irritation to programmers.

**8.Explain the significance of try-catch blocks?**

Whenever the exception occurs in Java, we need a way to tell the JVM what code to execute. To do this, we use the try and catch keywords. The try is used to define a block of code in which exceptions may occur. One or more catch clauses match a specific exception to a block of code that handles it.

  
  
**9.What is the use of finally block?**

The finally block encloses code that is always executed at some point after the try block, whether an exception was thrown or not. This is right place to close files, release your network sockets, connections, and perform any other cleanup your code requires.

**Note**: If the try block executes with no exceptions, the finally block is executed immediately after the try block completes. It there was an exception thrown, the finally block executes immediately after the proper catch block completes

**10.What if there is a break or return statement in try block followed by finally block?**

If there is a return statement in the try block, the finally block executes right after the return statement encountered, and before the return executes.

**Can we have the try block without catch block?**

Yes, we can have the try block without catch block, but finally block should follow the try block.  
**Note:** It is not valid to use a try clause without either a catch clause or a finally clause.

**12.What is the difference throw and throws?**

**throws:** Used in a method's signature if a method is capable of causing an exception that it does not handle, so that callers of the method can guard themselves against that exception. If a method is declared as throwing a particular class of exceptions, then any other method that calls it must either have a try-catch clause to handle that exception or must be declared to throw that exception (or its superclass) itself.

A method that does not handle an exception it throws has to announce this:

public void myfunc(int arg) **throws** MyException {  
        …  
    }

**throw:** Used to trigger an exception. The exception will be caught by the nearest try-catch clause that can catch that type of exception. The flow of execution stops immediately after the throw statement; any subsequent statements are not executed.

To throw an user-defined exception within a block, we use the throw command:

**throw** new MyException("I always wanted to throw an exception!");

**13.How to create custom exceptions?**

A. By extending the Exception class or one of its subclasses.

**Example:**

class MyException extends Exception {  
  public MyException() { super(); }  
  public MyException(String s) { super(s); }  
  }

**14.What are the different ways to handle exceptions?**

There are two ways to handle exceptions:

* Wrapping the desired code in a try block followed by a catch block to catch the exceptions.
* List the desired exceptions in the throws clause of the method and let the caller of the method handle those exceptions.

Q.If I write return at the end of the try block, will the finally block still execute?

Yes even if you write return as the last statement in the try block and no exception occurs, the finally block will execute. The finally block will execute and then the control return

Q.Is it necessary that each try block must be followed by a catch block?

It is not necessary that each try block must be followed by a catch block. It should be followed by either a catch block OR a finally block. And whatever exceptions are likely to be thrown should be declared in the throws clause of the method

Q.What is the basic difference between the 2 approaches to exception handling.   
1> try catch block and   
2> specifying the candidate exceptions in the throws clause?   
When should you use which approach?

In the first approach as a programmer of the method, you urself are dealing with the exception. This is fine if you are in a best position to decide should be done in case of an exception. Whereas if it is not the responsibility of the method to deal with it's own exceptions, then do not use this approach. In this case use the second approach.   
In the second approach we are forcing the caller of the method to catch the exceptions, that the method is likely to throw. This is often the approach library creators use. They list the exception in the throws clause and we must catch them.

Q.What are the different ways to handle exceptions?

There are two ways to handle exceptions,   
1. By wrapping the desired code in a try block followed by a catch block to catch the exceptions. and   
2. List the desired exceptions in the throws clause of the method and let the caller of the method hadle those exceptions.

Q.What is the difference between error and an exception?

An error is an irrecoverable condition occurring at runtime. Such as OutOfMemory error. These JVM errors and you can not repair them at runtime. While exceptions are conditions that occur because of bad input etc. e.g. FileNotFoundException will be thrown if the specified file does not exist. Or a NullPointerException will take place if you try using a null reference. In most of the cases it is possible to recover from an exception (probably by giving user a feedback for entering proper values etc.).

Q.What are Checked and UnChecked Exception?

A checked exception is some subclass of Exception (or Exception itself), excluding class RuntimeException and its subclasses.   
Making an exception checked forces client programmers to deal with the possibility that the exception will be thrown. eg, IOException thrown by java.io.FileInputStream's read() method·   
Unchecked exceptions are RuntimeException and any of its subclasses. Class Error and its subclasses also are unchecked. With an unchecked exception, however, the compiler doesn't force client programmers either to catch the   
exception or declare it in a throws clause. In fact, client programmers may not even know that the exception could be thrown. eg, StringIndexOutOfBoundsException thrown by String's charAt() method· Checked exceptions must be caught at compile time. Runtime exceptions do not need to be. Errors often cannot be

**Question:** What is user-defined exception in java ?

**Answer:** User-defined expections are the exceptions defined by the application developer which are errors related to specific [application](http://www.roseindia.net/interviewquestions/corejava/abstract-method.shtml). Application Developer can define the user defined exception by inherite the Exception class as shown below. Using this class we can throw new exceptions.

***Java Example :*** public class noFundException extends Exception { } Throw an exception using a throw statement: public class Fund { ... public Object getFunds() throws noFundException { if (Empty()) **throw** new noFundException(); ... } } User-defined exceptions should usually be checked.

**Question:** What is the difference between checked and Unchecked Exceptions in Java ?

**Answer:** All predefined exceptions in Java are either a checked exception or an unchecked exception. Checked exceptions must be caught using try .. catch() block or we should throw the exception using throws clause. If you dont, compilation of program will fail.

***Java Exception Hierarchy*** +--------+ | Object | +--------+ | | +-----------+ | Throwable | +-----------+ / \ / \ +-------+ +-----------+ | Error | | Exception | +-------+ +-----------+ / | \ / | \ \\_\_\_\_\_\_\_\_/ \\_\_\_\_\_\_/ \ +------------------+ unchecked checked | RuntimeException | +------------------+ / | | \ \\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ unchecked

Difference b/w java1.5 and java 1.6?

jdk 1.5 does not support the pointers and increment and decrement operators  
jdk 1.6 fully support the pointers and operators.

The new language features all have one thing in common: they take some common idiom and provide linguistic support for it.   
  
In other words, they shift the responsibility for writing the boilerplate code from the programmer to the compiler. Because the source code is now free of this boilerplate, it's easier to write and read.   
  
Because the compiler, unlike the programmer, never makes mistakes, the resulting code is also more likely to be free of bugs.   
  
The following are the features added to jdk 1.5

1. **Generics** - Provides compile-time type safety for collections and eliminates the drudgery of casting. Eg: you used to write (jdk 1.4)

List words = new ArrayList();  
You'll have to say: (jdk 1.5)

List<String> words = new ArrayList<String>();

1. **Enhanced for loop** - Eliminates the drudgery and error-proneness of iterators.
2. **Autoboxing/unboxing** - Eliminates the drudgery of manual conversion between primitive types (such as int) and wrapper types (such as Integer).
3. **Typesafe enums** - Provides all the well-known benefits of the Typesafe Enum pattern (*Effective Java*, Item 21) without the verbosity and the error-proneness.
4. **Static import** - Lets you avoid qualifying static members with class names, without the shortcomings of the Constant Interface antipattern (*Effective Java*, Item 17).
5. **Metadata** - Lets you avoid writing boilerplate code, by enabling tools to generate it from annotations in the source code. This leads to a "declarative" programming style where the programmer says *what* should be done and tools emit the code to do it.

### **How to retrieve a key and Value from a map ?**

Q. What is difference between shallow copy and deep copy ?

Only instances of classes that implement the Cloneable interface can be cloned. Trying to clone an object that does not implement the Cloneable interface throws a CloneNotSupportedException. Both shallow copy and deep copy object nedd to implement Cloneable Interface.   
  
Shallow copy :   
  
The java.lang.Object root superclass defines a clone() method.   
default behavior of clone() is to return a shallow copy of the object. This means that the values of all of the original object?s fields are copied to the fields of the new object.   
If a shallow copy is performed on obj1, then it is copied but its contained objects are not.   
For Example:   
Public class Emp   
{   
Private Address address;   
}   
Emp emp1 = new Emp();   
Address add = new Address();   
emp1. address= add.   
  
If we clone   
Emp emp2 = emp1.clone();   
Then emp2.addess reference to the same Address object which emp1 refer.   
  
  
Deep Copy :   
  
A deep copy occurs when an object is copied along with the objects to which it refers.   
A deep copy makes a distinct copy of each of the object?s fields, recursing through the entire graph of other objects referenced by the object being copied. The Java API provides no deep-copy equivalent to Object.clone(). One solution is to simply implement your own custom method (e.g., deepCopy()) that returns a deep copy of an instance of one of your classes.   
Your Object Class should implements Serializable and Cloneable interface. .   
public static Object copy(Object orig) {   
  
Object obj = null;   
  
try {   
  
// Write the object out to a byte array   
  
ByteArrayOutputStream bo = new ByteArrayOutputStream();   
  
ObjectOutputStream out = new ObjectOutputStream(bo);   
  
out.writeObject(orig);   
  
out.flush();   
  
out.close();   
  
// Make an input stream from the byte array and read   
  
// a copy of the object back in.   
  
ObjectInputStream in = new ObjectInputStream(   
  
new ByteArrayInputStream(bos.toByteArray()));   
  
obj = in.readObject();   
  
}   
  
catch(IOException e) {   
  
e.printStackTrace();   
  
}   
  
catch(ClassNotFoundException cnfe) {   
  
cnfe.printStackTrace();   
  
}   
  
return obj;   
  
}

Q.Why to override equals() and hashCode()? and How i can implement both equals() and hashCode() for Set ?

If you are implementing HashSet to store unique object then you need to implement equals() and hashcode() method.   
  
if two objects are equal according to the equals() method, they must have the same hashCode() value (although the reverse is not generally true).   
  
Two scenarios   
  
Case 1) : If you don't implement equals() and hashcode() method :   
  
When you are adding objects to HashSet , HashSet checks for uniqueness using   
equals() and hashcode() method the class ( ex. Emp class). If there is no equals() and hashcode() method the Emp class then HashSet checks default Object classes   
equals() and hashcode() method.   
  
In the Object class , equals method is   
public boolean equals(Object obj) {   
    return (this == obj);   
}   
Under this default implementation, two references are equal only if they refer to the exact same object. Similarly, the default implementation of hashCode() provided by Object is derived by mapping the memory address of the object to an integer value.   
  
This will fail to check if two Emp object with same employee name .   
For Example :   
Emp emp1 = new Emp();   
emp1.setName("sat");   
Emp emp2 = new Emp();   
emp2.setName("sat");   
Both the objects are same but based on the default above method both objects are dirrefent because references and hashcode are different .   
So in the HashSet you can find the duplicate emp objects.   
  
To overcome the issue equals() and hashcode() method need to override.   
Case 2) : If you override equals() and hashcode() method   
  
Example : implement equals and hashcode   
public class Emp {   
    private long empId;   
    String name = "";   
public boolean equals(Object o) {   
        if (o == this)   
        return true;   
        if (!(o instanceof Emp))   
        return false;   
        Emp emp = (Emp)o;   
        return emp. empId == empId &&   
        emp. name == name;   
           
        }   
       
    public int hashCode(){   
        int result = 10;   
        result = result \* new Integer(String.valueOf(empId)).intValue();   
        return result;   
    }   
}   
In the equals() , it check for name is same or not. This way you can find out the objects are equals or not.   
In the hashCode() also it return some unique value for each object. In this way if two Emp object has same empId then it will say both are same object.   
Now HashSet store only unique objects.   
If you do   
Emp emp1 = new Emp();   
emp1.setName("sat");   
Emp emp2 = new Emp();   
emp2.setName("sat");   
  
if(emp1.equals(emp2)){   
            System.out.println("equal");   
        }   
This will print : equal

Q.How to sort list of objects ( User Defined) using comparator? in Descending Order.

You can use Collections.sort(List,Comparator) to sort objects.   
Example : You have User Bean , you want to sort based on username filed.   
User Class :   
public class User {   
String userName = "";   
String city = "";   
String state = "";   
/\*\*     \* @return Returns the userName.   
     \*/   
    public String getUserName() {   
        return userName;   
    }   
    /\*\*   
     \* @param userName The userName to set.   
     \*/   
    public void setUserName(String userName) {   
        this.userName = userName;   
    }   
}   
  
Then you have to write a comparator class.   
public class UserNameComparator implements Comparator {   
     public int compare(Object user, Object anotherUser) {   
       
     String firstName1 = ((User) user).getUserName().toUpperCase();   
       
     String firstName2 = ((User) anotherUser).getUserName().toUpperCase();   
       
     return firstName1.compareTo(firstName2);   
       
     }   
    }   
  
Now sorting code   
User user1 = new User();   
        user1.setUserName("das");   
        User user2 = new User();   
        user2.setUserName("nick");   
        User user3 = new User();   
        user3.setUserName("ram");   
        User user4 = new User();   
        user4.setUserName("jadu");   
           
        ArrayList list = new ArrayList();   
        list.add(user1);   
        list.add(user2);   
        list.add(user3);   
        list.add(user4);   
           
        Collections.sort(list,new UserNameComparator()); // sort ascending order.   
  
// Descending order   
Collections.reverse(list);   
           
        for(int i=0;i<list.size();i++){   
            User usr = (User)list.get(i);   
            System.out.println(usr.getUserName());   
        }

What is the difference between int and Interger?

int is primitive type and Integer is Wrapper Class.   
For Example :   
int i=2;   
Integer i = new Integer (2);   
  
You can have question like why we need Wrapper Class(Integer )?   
  
Answer is , if you are using Collection objects like Hashtable etc. you can't use int as a key .you have to use object so you can use Integer class.   
  
Hashtable ht = new Hashtable ();   
ht.put(2, "USA") ;// You can not do this   
  
ht.put(new Integer(2), "USA") ;// you have to do like this.   
  
Singleton Double-checked locking in Java?

public static Singleton getInstance()   
{   
if (instance == null)   
{   
synchronized(Singleton.class) { //1   
if (instance == null) //2   
instance = new Singleton(); //3   
}   
}   
return instance;   
}   
Thread 1 enters the getInstance() method.   
  
  
Thread 1 enters the synchronized block at //1 because instance is null.   
  
  
Thread 1 is preempted by thread 2.   
  
  
Thread 2 enters the getInstance() method.   
  
  
Thread 2 attempts to acquire the lock at //1 because instance is still null. However, because thread 1 holds the lock, thread 2 blocks at //1.   
  
  
Thread 2 is preempted by thread 1.   
  
  
Thread 1 executes and because instance is still null at //2, creates a Singleton object and assigns its reference to instance.   
  
  
Thread 1 exits the synchronized block and returns instance from the getInstance() method.   
  
  
Thread 1 is preempted by thread 2.   
  
  
Thread 2 acquires the lock at //1 and checks to see if instance is null.   
  
  
Because instance is non-null, a second Singleton object is not created and the one created by thread 1 is returned.

What are the parameters to follow Creating and Destroying Objects in Java?

Item 1: Consider providing static factory methods instead of   
constructors   
public static Boolean valueOf(boolean b) {   
return (b ? Boolean.TRUE : Boolean.FALSE);   
}   
advantage of static factory methods is that, unlike constructors, they are not   
required to create a new object each time they're invoked.   
This allows immutable classes   
(Item 13) to use preconstructed instances or to cache instances as they're constructed and to   
dispense these instances repeatedly so as to avoid creating unnecessary duplicate objects.   
The Boolean.valueOf(boolean) method illustrates this technique: It never creates an object.   
This technique can greatly improve performance if equivalent objects are requested   
frequently, especially if these objects are expensive to create.   
it allows an immutable class to ensure that no two equal instances exist:   
a.equals(b) if and only if a==b. If a class makes this guarantee, then its clients can use   
the == operator instead of the equals(Object) method, which may result in a substantial   
performance improvement   
implements this   
optimization, and the String.intern method implements it in a limited form.   
  
advantage of static factory methods is that, unlike constructors, they can return   
an object of any subtype of their return type. This gives you great flexibility in choosing   
the class of the returned object.   
One application of this flexibility is that an API can return objects without making their   
classes public. Hiding implementation classes in this fashion can lead to a very compact API.   
  
Item 2: Enforce the singleton property with a private constructor   
  
Item 4: Avoid creating duplicate objects   
It is often appropriate to reuse a single object instead of creating a new functionally equivalent   
object each time it is needed. Reuse can be both faster and more stylish. An object can always   
be reused if it is immutable   
As an extreme example of what not to do, consider this statement:   
String s = new String("silly"); // DON'T DO THIS!   
The statement creates a new String instance each time it is executed, and none of those   
object creations is necessary. The argument to the String constructor ("silly") is itself a   
String instance, functionally identical to all of the objects created by the constructor. If this   
usage occurs in a loop or in a frequently invoked method, millions of String instances can be   
created needlessly.   
The improved version is simply the following:   
String s = "No longer silly";   
Item 5: Eliminate obsolete object references   
So where is the memory leak? If a stack grows and then shrinks, the objects that were popped   
off the stack will not be garbage collected, even if the program using the stack has no more   
references to them. This is because the stack maintains obsolete references to these objects.   
An obsolete reference is simply a reference that will never be dereferenced again. In this case,   
any references outside of the ?active portion? of the element array are obsolete. The active   
portion consists of the elements whose index is less than size.   
  
public Object pop() {   
if (size == 0)   
throw new EmptyStackException();   
return elements[--size];   
}   
The fix for this sort of problem is simple: Merely null out references once they become   
obsolete. In the case of our Stack class, the reference to an item becomes obsolete as soon as   
it's popped off the stack. The corrected version of the pop method looks like this:   
public Object pop() {   
if (size==0)   
throw new EmptyStackException();   
Object result = elements[--size];   
elements[size] = null; // Eliminate obsolete reference   
return result;   
}   
Item 6: Avoid finalizers   
There is no guarantee that finalizers will be executed promptly. It can take   
arbitrarily long between the time that an object becomes unreachable and the time that its   
finalizer is executed. This means that nothing time-critical should ever be done by a   
finalizer. For example, it is a grave error to depend on a finalizer to close open files because   
open file descriptors are a limited resource. If many files are left open because the JVM is   
tardy in executing finalizers, a program may fail because it can no longer open files.

Q.What are the steps in the JDBC connection?

While making a JDBC connection we go through the following steps :   
  
Step 1 : Register the database driver by using :   
Class.forName(\" driver classs for that specific database\" );   
  
Class.forName("com.mysql.jdbc.Driver");   
  
Step 2 : Now create a database connection using :   
  
Connection con = DriverManager.getConnection(url,username,password);   
  
Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/art",art,art);   
  
  
  
Step 3: Now Create a query using :   
  
Statement stmt = Connection.Statement(\"select \* from EMP\");   
  
Step 4 : Exceute the query :   
  
ResultSet rs = stmt.exceuteQuery();   
  
while(rs.next()){   
System.out.println(rs.getString(1));   
}

**Question:** Explain the user defined Exceptions?  
**Answer:** User defined Exceptions are the separate Exception classes defined by the user for specific purposed. An user defined can created by simply sub-classing it to the Exception class. This allows custom exceptions to be generated (using throw) and caught in the same way as normal exceptions.   
Example:  
class myCustomException extends Exception {  
     // The class simply has to exist to be an exception   
}

**Question:** Explain the new Features of JDBC 2.0 Core API?  
**Answer:** The JDBC 2.0 API includes the complete JDBC API, which includes both core and Optional Package API, and provides inductrial-strength database computing capabilities.   
New Features in JDBC 2.0 Core API:

* Scrollable result sets- using new methods in the ResultSet interface allows programmatically move the to particular row or to a position relative to its current position
* JDBC 2.0 Core API provides the Batch Updates functionality to the [java applications](http://www.roseindia.net/interviewquestions/corejava.shtml).
* Java applications can now use the ResultSet.updateXXX methods.
* New data types - interfaces mapping the SQL3 data types
* Custom  mapping of user-defined types (UTDs)
* Miscellaneous features, including performance hints, the use of character streams, full precision for java.math.BigDecimal values, additional [security](http://www.roseindia.net/interviewquestions/corejava.shtml), and support for time zones in date, time, and timestamp values.

**Question:** Explain garbage collection?  
**Answer:** Garbage collection is one of the most important feature of Java. Garbage collection is also called automatic memory management as JVM automatically removes the unused variables/objects (value is null) from the memory. User program cann't directly free the object from memory, instead it is the job of the garbage collector to automatically free the objects that are no longer referenced by a program. Every class inherits **finalize()** method from **java.lang.Object**, the finalize() method is called by garbage collector when it determines no more references to the object exists. In Java, it is good idea to explicitly assign **null** into a variable when no more in use. I Java on calling **System.gc()** and **Runtime.gc(),** JVM tries to recycle the unused objects, but there is no guarantee when all the objects will garbage collected. 

**Question:** How you can force the garbage collection?  
**Answer:** Garbage collection automatic process and can't be forced.  

**Java1.5 Features**

In this article, we'll discuss several of the new language features of JDK 1.5, including:

* **Generics—**Provides compile-time type safety for collections and eliminates the need for casting every time you get an object out of *Collections*.
* **Enhanced *For* loop—**Eliminates error-proneness of iterators.
* **Autoboxing/unboxing—**Eliminates need of manual conversion between primitive types (such as *double*) and wrapper types (such as *Double*).
* **Typesafe *enums*—**Provides all benefits of the Typesafe *enum* pattern.
* **Static import—**Eliminates the need for using class names prior to using the static member variables of other classes. This will make the code a bit neater.
* **Metadata—**Allows programmers to avoid writing boiler plate code and gives the opportunity for declarative programming.

Let's discuss each feature in detail and take a look at some examples.  
  
**Generics**Generics is one of the coolest features of JDK 1.5. By introducing [generics](http://builder.com.com/5100-6370-5074715.html), we will have compile-time type safety and possibly fewer *ClassCastExceptions* during run time. In JDK 1.5, you can declare the type of objects one collection will accept/return. In JDK 1.4, creating a List of employee names requires a collection object like the following statement:  
List listOfEmployeeName = new ArrayList();  
  
In JDK 1.5, you would use this statement:  
List<String> listOfEmployeeName = new ArrayList<String>();  
  
The cool part is that if you try to insert something that's not a string, you will find out at compile time and then you can fix the problem. Without generics, you discover such a bug when your customer calls and tells you that the program you shipped crashed with a *ClassCastException*.  
  
The other cool thing is that you don't have to cast when you get an element out of the collection. So instead of this type of statement:  
String employeeName = ((String) listOfEmployee.get(i));  
  
It's simply:  
String employeeName = listOfEmployee.get(i);  
  
Casting objects without knowing the type of object is not good, and more importantly, it can fail at run time. Suppose the user accidentally passes in a collection that contains string buffers rather than strings. In **Listing A**,

|  |
| --- |
| Listing A |
|  |
|  |
| //Client is required to pass collection of String and name  static boolean checkName(Collection employeeNameList, String name) {         for (Iterator i = employeeNamList.iterator(); i.hasNext(); ) {                 String s = (String) i.next();           if(s.equals(name)){                     return true;                           //print employee name here ......           }         }         return false;  } |

the client is required to pass in a collection of strings that the compiler can't enforce.**Listing B**

|  |
| --- |
| Listing B |
|  |
|  |
| static boolean checkName(**Collection<String> employeeNameList**, String name) {         for (Iterator i = employeeNamList.iterator(); i.hasNext(); ) {    **if(i.next().equals(name)){**                     return true;                          //print employee name here ......           }         }         return false;  } |

shows how the same method looks with generics.  
  
Now it's clear from the method signature that the input collection must contain only strings. If the client tries to pass in a collection of string buffers, the program won't compile. And notice that the method doesn't contain any casts. It's one line shorter and, once you get used to reading generics, it's clearer too.  
  
**Enhanced *For* Loop**Here's the syntax for the *For* loop in the current version of the JDK:  
void printAll(Collection c) {  
       for (Iterator i = c.iterator(); i.hasNext(); ) {  
               Employee emp = (Employee)i.next();  
               System.out.println(emp.getName());  
       }  
}  
  
Now here's the same method with an enhanced *For* statement:  
void printAll(Collection c) {  
       for (Object o : c)  
               System.out.println((TimerTask)o).getName());  
}  
  
In this *For* loop, you should read the ":" as "in," so the example reads "for Object o in c". You can see this *For* loop has more readability.  
 **Autoboxing and unboxing**  
In Java, we have primitive data types and wrapper classes around these primitive types. Most often programmers need to convert one type to another. Take a look at the code snippet in **Listing C**.

public class Employee {

       private static final Integer CHILD = new Integer(0);

       public static void main(String args[]) {

         //code for adding n to an Integer

         int n=10;

         Integer age= new Integer(30);

**Integer ageAfterTenYear= new Integer(age.intValue +10);**

       }

}

Notice how messy the inner-loop code that calculates *ageAfterTenYear* looks. Now take a look at the same program rewritten with autoboxing, as shown in **Listing D**.

|  |
| --- |
| Listing D |
|  |
|  |
| public class Employee {         public static void main(String args[]) {           int n=10;           Integer age= new Integer(30);           Integer ageAfterTenYear= age +10;         }  } |

One thing worth noting: Previously, if you unboxed Null, it became zero. In this code, the compiler would automatically convert Integer to *int* and add 10 to it, then convert that back to Integer.  
  
Typesafe *enums*  
Typesafe *enums* provide the following features:

* They provide compile-time type safety.
* They are objects, so you can put them in collections.
* They are implemented as a class, so you can add some methods.
* They provide a proper name space for the enumerated type.
* Their printed values are informative—if you print an *int enum,* you just see a number, which may not be that informative.

**Example 1:**  
enum Season { winter, spring, summer, fall }  
  
**Example 2:**  
public enum Coin {  
       penny(1), nickel(5), dime(10), quarter(25);  
  
       Coin(int value) { this.value = value; }  
  
       private final int value;  
  
       public int value() { return value; }  
}  
  
Static imports  
Static imports make code more readable. Currently, you use constants defined in other classes, like this:  
import org.yyy.pkg.Increment;  
  
class Employee {  
       public Double calculateSalary(Double salary{  
              return salary + Increment.INCREMENT \* salary;  
       }  
}  
  
But with static import, we can use those constants without providing the name of the class prior to constant name, like this:  
import static org.yyy.pkg.Increment;  
  
class Employee {  
       public Double calculateSalary(Double salary{  
               return salary + INCREMENT \* salary;  
       }  
}  
  
Note that we are able to call the INCREMENT constant without using the class name *Increment*.  
  
Metadata  
The metadata feature is focused on making a developer's life simpler with the support of tools provided by vendors. Take a look at the code in **Listing E**.

|  |
| --- |
| Listing E |
|  |
|  |
| import org.yyy.hr;      public interface EmployeeI extends Java.rmi.Remote {         public String getName()                 throws Java.rmi.RemoteException;         public String getLocation ()                 throws Java.rmi.RemoteException;  }      public class EmployeeImpl implements EmployeeI {         public String getName(){           // ......         }         public String getLocation (){               // ......         }  } |

With metadata support, you can write the code in **Listing E** like this:  
import org.yyy.hr;  
  
public class Employee {  
       @Remote public String getName() {  
               ...  
       }  
       @Remote public public String getLocation() {  
               ...  
       }  
}  
  
As you can see, all the boilerplate's code is gone.  
  
The new features and specifications that will be implemented in JDK 1.5 offer the Java programming community many more options for writing robust, scalable code. Serious Java programmers would do well to begin familiarizing themselves with the pending version of the Java programming language.

An annotation is the meta-tag that you will use in your code to give it some life. Annotation type is used for defining an annotation. You will use it when you want to create your own custom annotation. The type is the actual construct used, and the annotation is the specific usage of that type.

An annotation type definition takes an "at" (@) sign, followed by the interface keyword plus the annotation name. On the other hand, an annotation takes the form of an "at" sign (@), followed by the annotation type. This is simplest form of annotation. Additionally, you can put data within parenthesis after the annotation name. An example of each can be seen below:

#### **Example to Define an Annotation (Annotation type)**

public @interface MyAnnotation {

String doSomething();

}

#### **Example to Annotate Your Code (Annotation)**

MyAnnotation (doSomething="What to do")

public void mymethod() {

....

}

### **Annotation Types**

There are three annotation types:

* **Marker:** Marker type annotations have no elements, except the annotation name itself.

**Example:**

public @interface MyAnnotation {

}

**Usage:**

@MyAnnotation

public void mymethod() {

....

}

* **Single-Element:** Single-element, or single-value type, annotations provide a single piece of data only. This can be represented with a data=value pair or, simply with the value (a shortcut syntax) only, within parenthesis.

**Example:**

public @interface MyAnnotation

{

String doSomething();

}

**Usage:**

@MyAnnotation ("What to do")

public void mymethod() {

....

}

* **Full-value or multi-value:** Full-value type annotations have multiple data members. Therefore, you must use a full data=value parameter syntax for each member.

**Example:**

public @interface MyAnnotation {

String doSomething();

int count; String date();

}

**Usage:**

@MyAnnotation (doSomething="What to do", count=1,

date="09-09-2005")

public void mymethod() {

....

}

### **Rules of Thumb for Defining Annotation Type**

Here are some rules-of-thumb when defining an annotation type:

1. Annotation declaration should start with an 'at' sign like @, following with an interface keyword, following with the annotation name.
2. Method declarations should not have any parameters.
3. Method declarations should not have any throws clauses.
4. Return types of the method should be one of the following:
   * primitives
   * String
   * Class
   * enum
   * array of the above types

### **Annotations**

There are two types of annotations available with JDK5:

* **Simple annotations:** These are the basic types supplied with Tiger, which you can use to annotate your code only; you cannot use those to create a custom annotation type.
* **Meta annotations:** These are the annotation types designed for annotating annotation-type declarations. Simply speaking, these are called the annotations-of-annotations.

This tutorial introduces a new convenience feature of Java 5.0—the static import declaration—which enables programmers to refer to the imported static members of a class as if they were declared in the class that uses them. Thus, it is not necessary to qualify the imported static members with the name of the class in which they were declared. This tutorial is intended for students and professionals who are familiar with static class members in Java.

[Download the code for this tutorial here.](http://www.deitel.com/articles/java_tutorials/20060211/fig08_14.zip)

[Note: This tutorial is an excerpt (Section 8.12) of Chapter 8, Class and Objects: A Deeper Look, from our textbook [Java How to Program, 6/e](http://www.deitel.com/books/jHTP6/). This tutorial may refer to other chapters or sections of the book that are not included here. Permission Information: Deitel, Harvey M. and Paul J., JAVA HOW TO PROGRAM, ©2005, pp.380-382. Electronically reproduced by permission of Pearson Education, Inc., Upper Saddle River, New Jersey.]

# 8.12 static Import

In Section 6.3, you learned about the static fields and methods of class Math. We invoked class Math's static fields and methods by preceding each with the class name Math and a dot (.). A **static import** declaration (a new feature of J2SE 5.0) enables programmers to refer to imported static members as if they were declared in the class that uses them—the class name and a dot (.) are not required to use an imported static member.

    A static import declaration has two forms—one that imports a particular static member (which is known as **single static import**) and one that imports all static members of a class (which is known as **static import on demand**). The following syntax imports a particular static member:

import static *packageName*.*ClassName*.*staticMemberName*;

where *packageName* is the package of the class (e.g., java.lang), *ClassName* is the name of the class (e.g., Math) and *staticMemberName* is the name of the static field or method (e.g., PI or abs). The following syntax imports all static members of a class:

import static *packageName*.*ClassName*.\*;

where *packageName* is the package of the class (e.g., java.lang) and *ClassName* is the name of the class (e.g., Math). The asterisk (\*) indicates that *all* static members of the specified class should be available for use in the class(es) declared in the file. Note that static import declarations import only static class members. Regular import statements should be used to specify the classes used in a program.

     Figure 8.14 demonstrates a static import. Line 3 is a static import declaration, that imports all static fields and methods of class Math from package java.lang. Lines 9–12 access the Math class’s static field E (line 11) and the static methods sqrt (line 9), ceil (line 10), log (line 11) and cos (line 12) without preceding the field name or method names with class name Math and a dot.

|  |  |
| --- | --- |
| |  | | --- | | **Fig. 8.14** Static import Math methods. | |
| **1** // Fig. 8.14: StaticImportTest.java  **2** // Using static import to import static methods of class Math.  **3** import static java.lang.Math.\*;  **4**   **5** public class StaticImportTest   **6** {  **7** public static void main( String args[] )   **8** {  **9** System.out.printf( "sqrt( 900.0 ) = %.1f\n", sqrt( 900.0 ) );  **10** System.out.printf( "ceil( -9.8 ) = %.1f\n", ceil( -9.8 ) );  **11** System.out.printf( "log( E ) = %.1f\n", log( E ) );  **12** System.out.printf( "cos( 0.0 ) = %.1f\n", cos( 0.0 ) );  **13** } // end main  **14** } // end class StaticImportTest   |  | | --- | | sqrt( 900.0 ) = 30.0 ceil( -9.8 ) = -9.0 log( E ) = 1.0 cos( 0.0 ) = 1.0 | |