**CORE JAVA**

**Q. How are Observer and Observable used?**

**Ans.** Objects that subclass the Observable class maintain a list of observers. When an Observable object is updated it invokes the update() method of each of its observers to notify the observers that it has changed state. The Observer interface is implemented by objects that observe Observable objects.

**Q. What does strictfp mean?**

**Ans.** The strictfp modifier can be used with **top-level classes, nested classes, nested interfaces, and method declarations**. It can not be used with **variables, constructors, or initializer blocks**. It also cannot be combined with the abstract modifier.

If an expression is FP-strict, all intermediate values should fit into the range for float or double variables. If an expression is not FP-strict, there are no such restrictions.

A non-constant expression is considered FP-strict if any of the class, interface or method declarations that contain the expression have the strictfp modifier.

**Q. Is null a keyword?**

**Ans.** The null value is not a keyword. It is Reserved words for literal values.

There are three value :

false : A boolean literal value.

null : A reference literal value.

true : A boolean literal value

**Java Keyword:**

abstract continue for new switch

assert\*\*\* default goto\* package synchronized

boolean do if private this

break double implements protected throw

byte else import public throws

case enum\*\*\*\* instanceof return transient

catch extends int short try

char final interface static void

class finally long strictfp\*\* volatile

const\* float native super while

\* not used

\*\* added in 1.2

\*\*\* added in 1.4

\*\*\*\* added in 5.0

**Q. Which characters may be used as the second character of an identifier, but not as the first**

**character of an identifier?**

**Ans.** The digits 0 through 9 may not be used as the first character of an identifier but they may be used after the first character of an identifier.

**Q. How does Java handle integer overflows and underflows?**

**Ans.** It uses those low order bytes of the result that can fit into the size of the type allowed by the operation.

You could imagine that when you have only 2 places you are counting (so adding 1 each time)

00 01 10 11 100

But the last one gets cut down to "00" again. So there is your "overflow". You're back at 00. Now depending on what the bits mean, this can mean several things, but most of the time this means you are going from the highest value to the lowest. (11 to 00)

Mark peters adds a good one in the comments: even without overflow you'll have a problem, because the first bit is used as signing, so you'll go from high to low without losing that bit. You could say that the bit is 'separate' from the others.

Example:

If it overflows, it goes back to the [minimum value](http://docs.oracle.com/javase/6/docs/api/java/lang/Integer.html#MIN_VALUE) and continues from there. If it underflows, it goes back to the [maximum value](http://docs.oracle.com/javase/6/docs/api/java/lang/Integer.html#MAX_VALUE) and continues from there.

You can check that beforehand as follows:

public static boolean willAdditionOverflow(int left, int right) {

if (right < 0 && right != Integer.MIN\_VALUE) {

return willSubtractionOverflow(left, -right);

} else {

return (~(left ^ right) & (left ^ (left + right))) < 0;

}

}

public static boolean willSubtractionOverflow(int left, int right) {

if (right < 0) {

return willAdditionOverflow(left, -right);

} else {

return ((left ^ right) & (left ^ (left - right))) < 0;

}

}

**Q. What modifiers may be used with an inner class that is a member of an outer class?**

**Ans.**. A (non-local) inner class may be declared as public, protected, private, static, final, or abstract.

**Q. How many bits are used to represent Unicode, ASCII, UTF-16, and UTF-8 characters?**

**Ans.** 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.

**Q. Which java.util classes and interfaces support event handling?**

**Ans.** The EventObject class and the EventListener interface support event processing.

**Q. Is sizeof a keyword?**

**Ans.**. The sizeof operator is not a keyword.

Fallacy: Sizeof() is not needed because Java basic types' sizes are fixed.

**Q. Can an object's finalize() method be invoked while it is reachable?**

**Ans.** 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.

**Q. What is a native method?**

**Ans.** A native method is a method that is implemented in a language other than Java.

**Q. What are order of precedence and associativity, and how are they used?**

**Ans.** Order of precedence determines the order in which operators are evaluated in expressions. Associativity determines whether an expression is evaluated left-to-right or right-to-left

**Q. Can an anonymous class be declared as implementing an interface and extending a class?**

**Ans.** An anonymous class may implement an interface or extend a superclass, but may not be declared to do both.

**Anonymous Classes**

The concept of anonymous classes often confuses Java developers. Essentially,anonymous classes are limited-scope classes that are declared but are not given a name by the programmer (hence the designation "anonymous") and have only a limited lifespan. Although they can be used for a wide variety of applications, they are most commonly used as event handlers in GUI programs. **A common example of an anonymous class is shown in Example**

*class Base {*

*void method1() {}*

*void method2() {}*

*}*

*class A { // normal class*

*static class B {} // static nested class*

*class C {} // inner class*

*void f() {*

*class D {} // local inner class*

*}*

*void g() {*

*// anonymous class*

*Base bref = new Base() {*

*void method1() {}*

*};*

*}*

*}*

Because an anonymous class has no name, it cannot have an explicit constructor. Neither can an anonymous class be referred to outside its declaring expression, except indirectly through a superclass or interface object reference. Anonymous classes are never static, never abstract, and always final. Also, each declaration of an anonymous class is unique. For example, the following code declares two distinct anonymous classes:

*Base bref1 = new Base() {*

*void method1() {}*

*};*

*Base bref2 = new Base() {*

*void method1() {}*

*};*

**Each anonymous class is declared within an expression.**

**Q. What is the purpose of finalization?**

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

**Q. What is the difference between the Boolean & operator and the && operator?**

**Ans.** 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.

**Q. What is the purpose of the Runtime class?**

**Ans.** The purpose of the Runtime class is to provide access to the Java runtime system.

**Q. How many times may an object's finalize() method be invoked by the garbage collector?**

**Ans.** An object's finalize() method may only be invoked once by the garbage collector.

**Q. Which Java operator is right associative?**

**Ans.** The = operator is right associative.

**Q. What is the Locale class?**

**Ans.** The Locale class is used to tailor program output to the conventions of a particular geographic, political, or cultural region.

**Q. Can a double value be cast to a byte?**

**Ans.** Yes, a double value can be cast to a byte.

**Q. What must a class do to implement an interface?**

**Ans.** It must provide all of the methods in the interface and identify the interface in its implements clause.

**Q. How are Java source code files named?**

**Ans.**

1. A Java source code file takes the name of a public class or interface that is defined within the file.
2. A source code file may contain at most one public class or interface.
3. 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.
4. 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.
5. Source code files use the .java extension.

**Q. Can a Byte object be cast to a double value?**

**Ans.** No, an object cannot be cast to a primitive value.

**Q. What is the difference between a static and a nonstatic inner class?**

**Ans.** A nonstatic 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.

**Q. What is the Dictionary class?**

**Ans.** The Dictionary class provides the capability to store key-value pairs.

**Q. When can an object reference be cast to an interface reference?**

**Ans.** An object reference be cast to an interface reference when the object implements the referenced interface.

**Q. Which class is extended by all other classes?**

**Ans.** The Object class is extended by all other classes.

**Q. Can an object be garbage collected while it is still reachable?**

**Ans.** A reachable object cannot be garbage collected. Only unreachable objects may be garbage collected..

**Q. What is the difference between the Font and FontMetrics classes?**

**Ans.** The FontMetrics class is used to define implementation-specific properties, such as ascent and descent, of a Font object.

**Q. How is rounding performed under integer division?**

**Ans.** The fractional part of the result is truncated. This is known as rounding toward zero.

**Q. If a class is declared without any access modifiers, where may the class be accessed?**

**Ans.** 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.

**Q. What is the SimpleTimeZone class?**

**Ans.** The SimpleTimeZone class provides support for a Gregorian calendar.

**Q. Does a class inherit the constructors of its superclass?**

**Ans.**  A class does not inherit constructors from any of its superclasses.

**Q. What is the purpose of the System class?**

**Ans.** The purpose of the System class is to provide access to system resources.

**Q. Name the eight primitive Java types.**

**Ans.** The eight primitive types are byte, char, short, int, long, float, double, and boolean.

**Q. Which class should you use to obtain design information about an object?**

**Ans.** The Class class is used to obtain information about an object's design.

**Q. Is "abc" a primitive value?**

**Ans.** The String literal "abc" is not a primitive value. It is a String object.

**Q. What is the relationship between an event-listener interface and an event-adapter class?**

**Ans.** 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.

**Q. What restrictions are placed on the values of each case of a switch statement?**

**Ans.** During compilation, the values of each case of a switch statement must evaluate to a value that can be promoted to an int value.

**Q. What modifiers may be used with an interface declaration?**

**Ans.** An interface may be declared as public or abstract.

**Q. Is a class a subclass of itself?**

**Ans.**  A class is a subclass of itself.

**Q. What is the highest-level event class of the event-delegation model?**

**Ans.** The java.util.EventObject class is the highest-level class in the event-delegation class hierarchy.

**Q. What is the difference between a while statement and a do statement?**

**Ans.** 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.

**Q. What modifiers can be used with a local inner class?**

**Ans.** A local inner class may be **final or abstract**.

**Q. Which Math method is used to calculate the absolute value of a number?**

**Ans.** The abs() method is used to calculate absolute values.

**Q. When does the compiler supply a default constructor for a class?**

**Ans.** The compiler supplies a default constructor for a class if no other constructors are provided.

**Q. If a method is declared as protected, where may the method be accessed?**

**Ans. A** protected method may only be accessed by classes or interfaces of the same package or by subclasses of the class in which it is declared.

**Q. Which non-Unicode letter characters may be used as the first character of an identifier?**

**Ans.** The non-Unicode letter characters $ and \_ may appear as the first character of an identifier

**Q. What restrictions are placed on method overloading?**

**Ans.** Two methods may not have the same name and argument list but different return types.

**Q. What is casting?**

**Ans.** 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.

**Q. What is the return type of a program's main() method?**

**Ans.** A program's main() method has a void return type.

**Q. What class allows you to read objects directly from a stream?**

**Ans.** The ObjectInputStream class supports the reading of objects from input streams.

**Q. What is the difference between a field variable and a local variable?**

**Ans.** 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.

**Q. Under what conditions is an object's finalize() method invoked by the garbage collector?**

**Ans.** The garbage collector invokes an object's finalize() method when it detects that the object has become unreachable.

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

**Ans.** this() is used to invoke a constructor of the same class. super() is used to invoke a superclass constructor.

**Q. Why are the methods of the Math class static?**

**Ans.** So they can be invoked as if they are a mathematical code library.

**Q. What are the legal operands of the instanceof operator?**

**Ans.**. The left operand is an object reference or null value and the right operand is a class, interface, or array type.

**Q. If an object is garbage collected, can it become reachable again?**

**Ans.** Once an object is garbage collected, it ceases to exist. It can no longer become reachable again.

**Q. What are E and PI?**

**Ans.** E is the base of the natural logarithm and PI is mathematical value pi.

**Q. Are true and false keywords?**

**Ans.** The values true and false are not keywords.

**Q. What is a void return type?**

**Ans.** A void return type indicates that a method does not return a value.

**Q. What is the purpose of the enableEvents() method?**

**Ans.** 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.

**Q. What happens when you add a double value to a String?**

**Ans.** The result is a String object.

**Q. What is your platform's default character encoding?**

**Ans.** If you are running Java on English Windows platforms, it is probably Cp1252. If you are running Java on English Solaris platforms, it is most likely 8859\_1..

**Q. Which package is always imported by default?**

**Ans.** The java.lang package is always imported by default.

**Q. What interface must an object implement before it can be written to a stream as an object?**

**Ans.** An object must implement the Serializable or Externalizable interface before it can be written to a stream as an object.

**Q. How are this and super used?**

**Ans.** 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.

**Q. What is the purpose of garbage collection?**

**Ans.** 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.

**Q. What is a compilation unit?**

**Ans.** A compilation unit is a Java source code file.

**Q. What interface is extended by AWT event listeners?**

**Ans.** All AWT event listeners extend the java.util.EventListener interface.

**Q. What restrictions are placed on method overriding?**

**Ans.** 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.

**Q. What happens if an exception is not caught?**

**Ans.** 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.

**Q. Can an abstract class be final?**

**Ans.** An abstract class may not be declared as final.

**Q. What is the ResourceBundle class?**

**Ans.** 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.

**Q. What is numeric promotion?**

**Ans.** Numeric promotion is the conversion of a smaller numeric type to a larger numeric type, so that integer and floating-point operations 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.

**Q. What is the difference between a public and a non-public class?**

**Ans.** A public class may be accessed outside of its package. A non-public class may not be accessed outside of its package.

**Q. To what value is a variable of the boolean type automatically initialized?**

**Ans.** The default value of the boolean type is false.

**Q. What is the difference between the prefix and postfix forms of the ++ operator?**

**Ans.** The prefix form performs the increment operation and returns the value of the increment operation. The postfix form returns the current value all of the expression and then performs the increment operation on that value.

**Q. What is the purpose of a statement block?**

**Ans.** A statement block is used to organize a sequence of statements as a single statement group.

**Q. What is a Java package and how is it used?**

**Ans.** 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.

**Q. What modifiers may be used with a top-level class?**

**Ans.** A top-level class may be public, abstract, or final.

**Q. What are the Object and Class classes used for?**

**Ans.** The Object class is the highest-level class in the Java class hierarchy. The Class class is used to represent the classes and interfaces that are loaded by a Java program.

**Q. When is an object subject to garbage collection?**

**Ans.** An object is subject to garbage collection when it becomes unreachable to the program in which it is used.

**Q. What is the difference between an if statement and a switch statement?**

**Ans.** 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.

**Q. What is the purpose of the toolkit in the Abstract Window Toolkit (AWT)? How does AWT**

**work ?**

**Ans.** The AWT toolkit is an interface between the abstract window layer and a specific windowing

Implementation.

**Q. What is Java Beans ?**

**Ans.** According to JavaSoft, "A Java Bean is a reusable software component that can be manipulated visually in a builder tool."

**Q. What do you know about networking support in Java ?**

**Ans.** Java supports "low-level" and "high-level" classes. "Low-level" classes provide support for socket programming: Socket, DatagramSocket, and ServerSocket classes. "High-level" classes provide "Web programming": URL, URLEncoder, and URLConnection classes. Networking programming classes ease the programming of network applications, but do not substitute your knowledge of networking. Java networking like anything else in Java is platform-independent.

**Q. What is it reflection (introspection) ? Why is reflection possible in the Java language?**

**Ans.** Reflection (introspection) is querying a class about its properties, and operating on methods and

fields by the name for a given object instance. Reflection is possible in the Java language because of late binding.

**Q. Why are Java ARchive (JAR) files important?**

**Ans.** JAR files bundle .class files and optimize applet downloads.

**Q. Describe, in general, how java's garbage collector works?**

**Ans.** The Java runtime environment deletes objects when it determines that they are no longer being used. This process is known as garbage collection.

The Java runtime environment supports a garbage collector that periodically frees the memory used by objects that are no longer needed. The Java garbage collector is a mark-sweep garbage collector that scans Java's dynamic memory areas for objects, marking those that are referenced. After all possible paths to objects are investigated, those objects that are not marked (i.e. are not referenced) are known to be garbage and are collected.

**Q. How can you force all derived classes to implement a method present in the base class?**

**Ans.** Creating and implementing an interface would be the best way for this situation. Just create an interface with empty methods which forces a programmer to implement all the methods present under it. Another way of achieving this task is to declare a class as abstract with all its methods abstract.

**Q. What is the difference between an Applet and an Application?**

**Ans.** 1. Applets can be embedded in HTML pages and downloaded over the Internet whereas Applications have no special support in HTML for embedding or downloading.

2. Applets can only be executed inside a java compatible container, such as a browser or appletviewer whereas Applications are executed at command line by java.exe or jview.exe.

3. Applets execute under strict security limitations that disallow certain operations(sandbox model security) whereas Applications have no inherent security restrictions.

4. Applets don't have the main() method as in applications. Instead they operate on an entirely different mechanism where they are initialized by init(),started by start(),stopped by stop() or destroyed by destroy().

**Q. Java says "write once, run anywhere". What are some ways this isn't quite true?**

**Ans.** Any time you use system calls specific to one operating system and do not create alternative calls for another operating system, your program will not function correctly.

Solaris systems and Intel systems order the bits of an integer differently. (You may have heard of little

endian vs. big endian)

If your code uses bit shifting, or other binary operators, they will not work on systems that have opposite endianness.

**Q. Describe java's security model.**

**Ans.** Java's security model is one of the most interesting and unique aspects of the language. For the most part it's broken into two pieces: the user adjustable security manager that checks various API operations like file access, and the bytecode verifier that asserts the validity of compiled byte code.

public abstract class SecurityManager java.lang.SecurityManager is an abstract class which different

applications subclass to implement a particular security policy. It allows an application to determine

whether or not a particular operation will generate a security exception.

**Q. What are native methods? How do you use them?**

**Ans.** Native methods are methods that are defined as public static methods within a java class, but whose implementation is provided in another programming language such as C.

**Q. What is RMI?**

**Ans.** RMI stands for Remote Method Invocation. Traditional approaches to executing code on other machines across a network have been confusing as well as tedious and error-prone to implement. The nicest way to think about this problem is that some object happens to live on another machine, and that you can send a message to the remote object and get a result as if the object lived on your local machine. This simplification is exactly what Java Remote Method Invocation (RMI) allows you to do.

**Q. Does Java have "goto"?**

**Ans.** No

**Q. Are constructors inherited? Can a subclass call the parent's class constructor? When?**

**Ans.** You cannot inherit a constructor. That is, you cannot create a instance of a subclass using a constructor of one of its superclasses. One of the main reasons is because you probably don't want to override the superclass's constructor, which would be possible if they were inherited. By giving the developer the ability to override a superclass's constructor you would erode the encapsulation abilities of the language.

**Q. Does Java have destructors?**

**Ans.** No garbage collector does the job working in the background

**Q. What does the "abstract" keyword mean in front of a method? A class?**

**Ans.** Abstract keyword declares either a method or a class. If a method has a abstract keyword in front of it,it is called abstract method.Abstract method hs no body.It has only arguments and return type.Abstract methods act as placeholder methods that are implemented in the subclasses.

Abstract classes can't be instantiated.If a class is declared as abstract,no objects of that class can be

created.If a class contains any abstract method it must be declared as abstract

**Q. Name four methods every Java class will have.**

**Ans.** public String toString();

public Object clone();

public boolean equals();

public int hashCode();

**Q. What is the difference between instanceof and isInstance?**

**Ans.** instanceof is used to check to see if an object can be cast into a specified type without throwing a cast class exception.

isInstance()

Determines if the specified Object is assignment-compatible with the object represented by this Class. This method is the dynamic equivalent of the Java language instanceof operator. The method returns true if the specified Object argument is non-null and can be cast to the reference type represented by this Class object without raising a ClassCastException. It returns false otherwise.

**Q. How can I improve the performance of a java application, what are the java optimization**

**techniques.**

**Ans.** That question is about as meaningful as "how do you code stuff?" There's no one right answer, and the interviewer probably just wanted to see if you knew \*anything\* about optimization. The answer is to go to the Performance thread and read the postings. If you search there for "books", you should find some good references which will get you started. You should also search for "tools".

**Q. What is immutable object? Can you write immutable object?**

**Ans.** You need to make class final and all its member final so that once objects gets crated no one can modify its state. You can achieve same functionality by making member as non final but private and not modifying them except in constructor.  
  
**Q. Does all property of immutable object needs to be final?**  
**Ans.** Not necessary as stated above you can achieve same functionality by making member as non final but private and not modifying them except in constructor.

**Question: What do you understand by Synchronization?  
Ans.**Synchronization is a process of controlling the access of shared resources by the multiple threads in such a manner that only one thread can access one resource at a time. In non synchronized multithreaded application, it is possible for one thread to modify a shared object while another thread is in the process of using or updating the object's value. Synchronization prevents such type of data corruption.  
E.g. Synchronizing a function:  
public synchronized void Method1 () {  
   // Appropriate method-related code.   
}  
E.g. Synchronizing a block of code inside a function:  
public myFunction (){  
  synchronized (this) {   
  // Synchronized code here.  
   }  
}

**Q.** **Describe the wrapper classes in Java.**  
**Ans.** 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 |

**Q: What is the difference between the instanceof and getclass, these two are same or not ?  
Ans:**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 operator 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  
}  
}

**Q. What is difference between Path and Classpath?**

**Ans:** Path and Classpath are operating system level environment variables. Path is used define where the system can find the executables(.exe) files and classpath is used to specify the location .class files.

**Q. What are local variables?**

**Ans.** Local variables are those which are declared within a block of code like methods. Local variables should be initialised before accessing them.

**Q. Can a main() method be overloaded?**

**Ans.** Yes. You can have any number of main() methods with different method signature and implementation in the class.

**Q. Can a main() method be declared final?**

**Ans.** Yes. Any inheriting class will not be able to have it's own default main() method.

**Q. Does the order of public and static declaration matter in main() method?**

**Ans. No**. It doesn't matter but void should always come before main().

**Q. Can a source file contain more than one class declaration?**

**Ans.** Yes a single source file can contain any number of Class declarations but only one of the class can be declared as public.

**Q. Which package is imported by default?**

**Ans.** java.lang package is imported by default even without a package declaration.

**Q. Can a class declared as private be accessed outside it's package?**

**Ans.** Not possible.

**Q. Can you give few examples of final classes defined in Java API?**

**Ans**. java.lang.String, java.lang.Math are final classes.

**Q. How is final different from finally and finalize()?**

**Ans.** final is a modifier which can be applied to a class or a method or a variable. final class can't be inherited,final method can't be overridden and final variable can't be changed.   
finally is an exception handling code section which gets executed whether an exception is raised or not by the try block code segment.   
finalize() is a method of Object class which will be executed by the JVM just before garbage collecting object to give a final chance for resource releasing activity.

**Q. Can a class be declared as static?**

**Ans.** No a class cannot be defined as static. Only a method, a variable or a block of code can be declared as static.

**Q. What are the restriction imposed on a static method or a static block of code?**

**Ans.** A static method should not refer to instance variables without creating an instance and cannot use "this" operator to refer the instance.

**Q. I want to print "Hello" even before main() is executed. How will you achieve that?**

**Ans.** Print the statement inside a static block of code. Static blocks get executed when the class gets loaded into the memory and even before the creation of an object. Hence it will be executed before the main() method. And it will be executed only once.

**Q. What is the importance of static variable?**

**Ans.** static variables are class level variables where all objects of the class refer to the same variable. If one object changes the value then the change gets reflected in all the objects.

**Q. Can we declare a static variable inside a method?**

**Ans.** Static variables are class level variables and they can't be declared inside a method. If declared, the class will not compile.

**Q. Can a class be defined inside an Interface?**

**Ans.** Yes it's possible.

**Q. Can an Interface be defined inside a class?**

**Ans.** Yes it's possible.

**Q. Which object oriented Concept is achieved by using overloading and overriding?**

**Ans.** Polymorphism.

**Q. Why does Java not support operator overloading?**

**Ans.** Operator overloading makes the code very difficult to read and maintain. To maintain code simplicity, Java doesn't support operator overloading.

**Q. Can we define private and protected modifiers for variables in interfaces?**

**Ans.** No.

**Q. What value does read() return when it has reached the end of a file?**

**Ans.** The read() method returns -1 when it has reached the end of a file.

**Q. What is the difference between a static and a non-static inner class?**

**Ans.** 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.

**Q. What is an object's lock and which object's have locks?**

**Ans.** An object's lock is a mechanism that is used by multiple threads to obtain synchronized access to the object. A thread may execute a synchronized method of an object only after it has acquired the object's lock. All objects and classes have locks. A class's lock is acquired on the class's Class object.

**Q. When can an object reference be cast to an interface reference?**

**Ans.** An object reference be cast to an interface reference when the object implements the referenced interface.

**Q Which class is extended by all other classes?**

**Ans.** The Object class is extended by all other classes.

**Q. Which non-Unicode letter characters may be used as the first character of an identifier?**

**Ans.** The non-Unicode letter characters $ and \_ may appear as the first character of an identifier

**Q. What restrictions are placed on method overloading?**

**Ans.** Two methods may not have the same name and argument list but different return types.

**Q. What is casting?**

**Ans.** 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.

**Q. What is Downcasting ?**

**Ans.** Downcasting is the casting from a general to a more specific type, i.e. casting down the hierarchy.

**Q. What modifiers may be used with an inner class that is a member of an outer class?**

**Ans.** A (non-local) inner class may be declared as public, protected, private, static, final, or abstract.

**Q. How many bits are used to represent Unicode, ASCII, UTF-16, and UTF-8 characters?**

**Ans.** 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.

**Q. What is a native method?**

**Ans.** A native method is a method that is implemented in a language other than Java.

**Q. Can an anonymous class be declared as implementing an interface and extending a class?**

**Ans.** An anonymous class may implement an interface or extend a superclass, but may not be declared to do both.

**Q. Does a class inherit the constructors of its superclass?**

**Ans.** A class does not inherit constructors from any of its superclasses.

**Q. Name the eight primitive Java types.**

**Ans.** The eight primitive types are byte, char, short, int, long, float, double, and boolean.

**Q. What happens when you add a double value to a String?**

**Ans.** The result is a String object.

**Q. What is numeric promotion?**

**Ans.** Numeric promotion is the conversion of a smaller numeric type to a larger numeric type, so that integer and floating-point operations 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.

**Q. What restrictions are placed on method overriding?**

**Ans.** 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.

**Q. What are the practical benefits, if any, of importing a specific class rather than an entire package (e.g. import java.net.\* versus import java.net.Socket)?**

**Ans.** It makes no difference in the generated class files since only the classes that are actually used are referenced by the generated class file. There is another practical benefit to importing single classes, and this arises when two (or more) packages have classes with the same name. Take java.util.Timer and javax.swing.Timer, for example. If I import java.util.\* and javax.swing.\* and then try to use "Timer", I get an error while compiling (the class name is ambiguous between both packages). Let's say what you really wanted was the javax.swing.Timer class, and the only classes you plan on using in java.util are Collection and HashMap. In this case, some people will prefer to import java.util.Collection and import java.util.HashMap instead of importing java.util.\*. This will now allow them to use Timer, Collection, HashMap, and other javax.swing classes without using fully qualified class names in.

**Q. What is constructor chaining and how is it achieved in Java ?**

**Ans.** A child object constructor always first needs to construct its parent (which in turn calls its parent constructor.). In Java it is done via an implicit call to the no-args constructor as the first statement.

**Q. What is the difference between a break statement and a continue statement?**

**Ans.** 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.

**Q. How many types of memory areas are allocated by JVM?**

**Ans.** JVM (Java Virtual Machine) is an abstract machine.It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms (i.e.JVM is platform dependent).

The JVM performs four main tasks:

●Loads code

●Verifies code

●Executes code

●Provides runtime environment

JVM provides definitions for the:

●Memory area

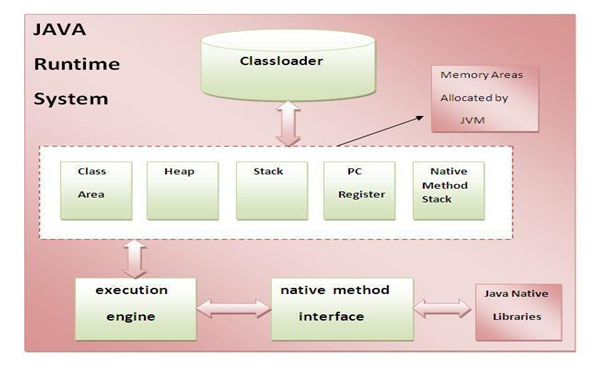
●Class file format

●Register set

●Garbage-collected heap

●Fatal error reporting etc.

Internal Architecture of JVM



**1) Classloader:** Classloader is a subsystem of JVM that is used to load class files.

**2) Class(Method) Area:**Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

**3) Heap:** It is the runtime data area in which objects are allocated.

**4) Stack:** Java Stack stores frames.It holds local variables and partial results, and plays a part in method invocation and return.

Each thread has a private JVM stack, created at the same time as thread.

A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

**5) Program Counter Regiser:**

PC (program counter) register. It contains the address of the Java virtual machine instruction currently being executed.

**6) Native Method Stack:**

It contains all the native methods used in the application.

**7) Execution Engine:**

It contains:

1)A virtual processor

2)Interpreter:Read bytecode stream then execute the instructions.

3)Just-In-Time(JIT) compiler:It is used to improve the performance.JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation.Here the term ?compiler? refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

**Q.What is classloader?**

**Ans.** The classloader is a subsystem of JVM that is used to load classes and interfaces.There are many types of classloaders e.g. Bootstrap classloader, Extension classloader, System classloader, Plugin classloader etc.

**Q. Is Empty .java file name a valid source file name?**

**Ans.** Yes, save your java file by .java only, compile it by javac .java and run by java yourclassname Let's take a simple example:

1. //save by .java only2.

3. class A{

4. public static void main(String args[]){

5. System.out.println("Hello java");

6. }

7. }

8.

9. //compile by javac .java

10. //run by java A

compile it by javac .java

run it by java A

**Q. What is difference between object oriented programming language and object based programming language?**

**Ans.** Object based programming languages follow all the features of OOPs except Inheritance. Examples of object based programming languages are JavaScript, VBScript etc.

**Q. Constructor Overloading**

**Ans.** Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists.The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

**Q. What is the difference between constructor and method ?**

|  |  |
| --- | --- |
| **Constructor** | **Method** |
| Constructor is used to initialize the state of an object. | Method is used to expose behaviour of an object. |
| Constructor must not have return type. | Method must have return type. |
| Constructor is invoked implicitly. | Method is invoked explicitly. |
| The java compiler provides a default constructor if you don't have any constructor. | Method is not provided by compiler in any case. |
| Constructor name must be same as the class name. | Method name may or may not be same as class name. |

**Q. Copying the values of one object to another like copy constructor in C++**

**Ans.** There are many ways to copy the values of one object into another. They are:

• By constructor

• By assigning the values of one object into another

• By clone() method of Object class

**Q. What is static variable?**

**Ans. static keyword**

The static keyword is used in java mainly for memory management. We may apply static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than instance of the class.

**The static can be:**

1.variable (also known as class variable)

2.method (also known as class method)

3.block

4.nested class

**1) static variable**

If you declare any variable as static, it is known static variable.

•The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees,college name of students etc.

•The static variable gets memory only once in class area at the time of class loading.

Advantage of static variable

It makes your program memory efficient (i.e it saves memory).

**Example:**

class Student{

int rollno;

String name;

static String college ="ITS";

Student(int r,String n){

rollno = r;

name = n;

}

void display (){System.out.println(rollno+" "+name+" "+college);}

public static void main(String args[]){

Student s1 = new Student (111,"Karan");

Student s2 = new Student (222,"Aryan");

s1.display();

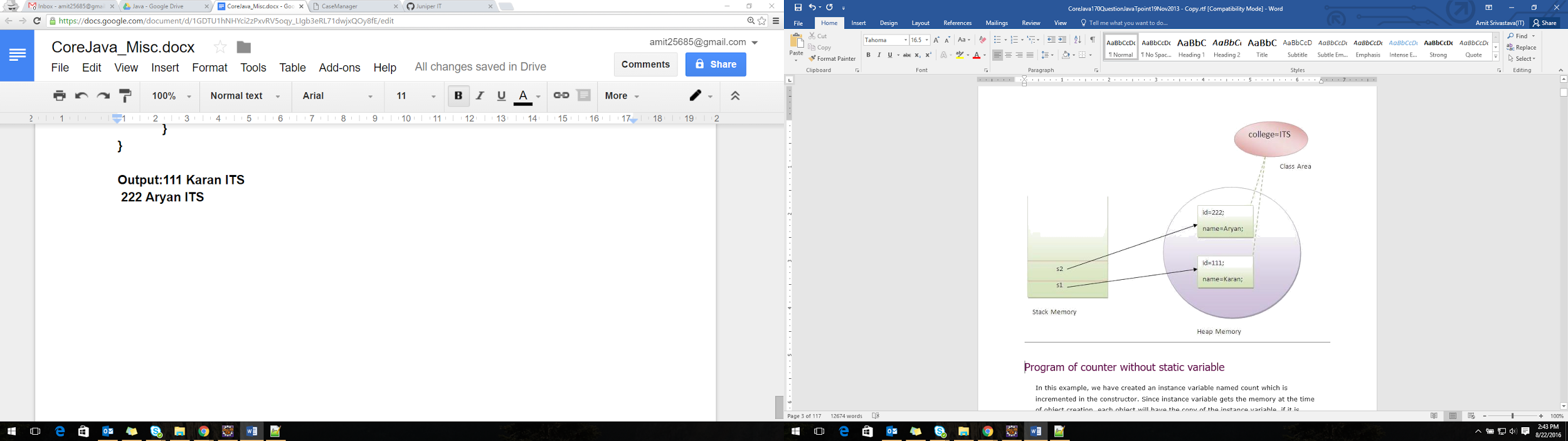
s2.display();

}

}

Output:111 Karan ITS

222 Aryan ITS



**Q. What is static method?**

**Ans.** If you apply static keyword with any method, it is known as static method.

• A static method belongs to the class rather than object of a class.

• A static method can be invoked without the need for creating an instance of a class.

• static method can access static data member and can change the value of it.

**Example of static method**

//Program of changing the common property of all objects(static field).

class Student{

int rollno;

String name;

static String college = "ITS";

static void change(){

college = "BBDIT";

}

Student(int r, String n){

rollno = r;

name = n;

}

void display (){System.out.println(rollno+" "+name+" "+college);}

public static void main(String args[]){

Student.change();

Student s1 = new Student (111,"Karan");

Student s2 = new Student (222,"Aryan");

Student s3 = new Student (333,"Sonoo");

s1.display();

s2.display();

s3.display();

}

}

Output:111 Karan BBDIT

222 Aryan BBDIT

333 Sonoo BBDIT

Restrictions for static method

There are two main restrictions for the static method. They are:

1. The static method can not use non static data member or call non-static method directly.

2. this and super cannot be used in static context.

**Q. Why main method is static?**

**Ans.** Because object is not required to call static method if it were non-static method, jvm creats object first then call main() method that will lead the problem of extra memory allocation.

**Q. What is static block?**

**Ans.** Is used to initialize the static data member.

•It is executed before main method at the time of classloading.

Example of static block

class A{

static{System.out.println("static block is invoked");}

public static void main(String args[]){

System.out.println("Hello main");

}

}

Output:static block is invoked

Hello main

**Q. Can we execute a program without main() method?**

**Ans.** Yes, one of the way is static block but in previous version of JDK not in JDK 1.7.

class A{

static{

System.out.println("static block is invoked");

System.exit(0);

}

}

**Q. What if the static modifier is removed from the signature of the main method?**

**Ans.** Program compiles. But at runtime throws an error "NoSuchMethodError".

**Q. What is difference between static (class) method and instance method?**

|  |  |
| --- | --- |
| **Static Method** | **Instance Method** |
| A method i.e. declared as static is known as static method. | A method i.e. not declared as static is known as instance method. |
| Object is not required to call static method. | Object is required to call instance methods. |
| 3)Non-static (instance) members cannot be accessed in static context (static method, static block and static nested class) directly. | static and non-static variables both can accessed in instance methods. |
| 4)For example: public static int cube(int n){ return n\*n\*n;} | For example: public void msg(){...}. |

**Q.What is this in java?**

**Ans.** There can be a lot of usage of this keyword. In java, this is a reference variable that refers to the current object.

Usage of this keyword

Here is given the 6 usage of this keyword.

1. this keyword can be used to refer current class instance variable.

2. this() can be used to invoke current class constructor.

3. this keyword can be used to invoke current class method (implicitly)

4. this can be passed as an argument in the method call.

5. this can be passed as argument in the constructor call.

6. this keyword can also be used to return the current class instance.

**Q. What is composition?**

**Ans.** Holding the reference of the other class within some other class is known as composition.

**Q. What is difference between aggregation and composition?**

**Ans.** Aggregation represents weak relationship whereas composition represents strong relationship. For example: bike has an indicator (aggregation) but bike has an engine (composition).

**Q. Why Java does not support pointers?**

**Ans.**  Pointer is a variable that refers to the memory address. They are not used in java because they are unsafe(unsecured) and complex to understand.

**Q. What is super in java?**

**Ans. super keyword**

The super is a reference variable that is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly i.e. referred by super reference variable.

Usage of super Keyword

1.super is used to refer immediate parent class instance variable.

2.super() is used to invoke immediate parent class constructor.

3.super is used to invoke immediate parent class method.

**Q. Can you use this() and super() both in a constructor?**

**Ans.** No. Because super() or this() must be the first statement.

**Q. What is object cloning?**

**Ans. Object Cloning in Java**

The object cloning is a way to create exact copy of an object. For this purpose, clone() method of Object class is used to clone an object. The java.lang.Cloneable interface must be implemented by the class whose object clone we want to create. If we don't implement Cloneable interface, clone() method generatesCloneNotSupportedException.

The clone() method is defined in the Object class. Syntax of the clone() method is as follows:

**1. protected Object clone() throws CloneNotSupportedException**

Why use clone() method ?

The clone() method saves the extra processing task for creating the exact copy of an object. If we perform it by using the new keyword, it will take a lot of processing to be performed that is why we use object cloning.

**Advantage of Object cloning**

Less processing task.

**Example of clone() method (Object cloning)**

**Let's see the simple example of object cloning**

class Student implements Cloneable{

int rollno;

String name;

tudent(int rollno,String name){

this.rollno=rollno;

this.name=name;

}

public Object clone()throws CloneNotSupportedException{

return super.clone();

}

public static void main(String args[]){

try{

Student s1=new Student(101,"amit");

Student s2=(Student)s1.clone();

System.out.println(s1.rollno+" "+s1.name);

System.out.println(s2.rollno+" "+s2.name);

}catch(CloneNotSupportedException c){}

}

}

Output:101 amit

101 amit

**Q. What is method overloading?**

**Ans. Method Overloading in Java**

If a class have multiple methods by same name but different parameters, it is known as Method Overloading. If we have to perform only one operation, having same name of the methods increases the readability of the program.Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behaviour of the method because its name differs. So, we perform method overloading to figure out the program quickly.

**Advantage of method overloading?**

Method overloading increases the readability of the program.

Different ways to overload the method

**There are two ways to overload the method in java**

1. By changing number of arguments

2. By changing the data type

**Note:-In java, Method Overloading is not possible by changing the return type of the method.**

**Q. Why Method Overloading is not possible by changing the return type of method?**

**Ans.** In java, method overloading is not possible by changing the return type of the method because there may occur ambiguity. Let's see how ambiguity may occur: because there was problem:

class Calculation{

int sum(int a,int b){System.out.println(a+b);}

double sum(int a,int b){System.out.println(a+b);}

public static void main(String args[]){

Calculation obj=new Calculation();

int result=obj.sum(20,20); //Compile Time Error

}

}

int result=obj.sum(20,20); //Here how can java determine which sum() method should be called

**Q. Can we overload main() method?**

**Ans.** Yes, by method overloading. You can have any number of main methods in a class by method overloading. Let's see the simple example:

class Simple{

public static void main(int a){

System.out.println(a);

}

public static void main(String args[]){

System.out.println("main() method invoked");

main(10);

}

}

**Output:**main() method invoked

10

**Q. What is method overriding:**

**Ans.** Method Overriding in Java

Having the same method in the subclass as declared in the parent class is known as method overriding. In other words, If subclass provides the specific implementation of the method i.e. already provided by its parent class, it is known as Method Overriding.

Advantage of Method Overriding

● Method Overriding is used to provide specific implementation of a method that is already provided by its super class.

● Method Overriding is used for Runtime Polymorphism

Rules for Method Overriding:

1. method must have same name as in the parent class

2. method must have same parameter as in the parent class.

3. must be inheritance (IS-A) relationship.

**Q. Can we override static method?**

**Ans.** No, you can't override the static method because they are the part of class not object.

**Q. Why we cannot override static method?**

**Ans.** It is because the static method is the part of class and it is bound with class whereas instance method is bound with object and static gets memory in class area and instance gets memory in heap.

**Q. Difference between method Overloading and Overriding.**

|  |  |
| --- | --- |
| **Method Overloading** | **Method Overriding** |
| Method overloading increases the readability of the program. | Method overriding provides the specific implementation of the method that is already provided by its superclass. |
| method overloading is occurs within the class. | Method overriding occurs in two classes that have IS-A relationship. |
| In this case, parameter must be different. | In this case, parameter must be same. |

**Q. Can you have virtual functions in Java?**

**Ans.** Yes, all functions in Java are virtual by default.

**Q. What is covariant return type?**

**Ans. Covariant Return Type**

The covariant return type specifies that the return type may vary in the same direction as the subclass. Before Java5, it was not possible to override any method by changing the return type. But now, since Java5, it is possible to override method by changing the return type if subclass overrides any method whose return type is Non-Primitive but it changes its return type to subclass type. Let's take a simple example:

class A{

A get(){return this;}

}

class B extends A{

B get(){return this;}

void message(){System.out.println("welcome to covariant return type");}

public static void main(String args[]){

new B().get().message();

}

}

Output:welcome to covariant return type

As you can see in the above example, the return type of the get() method of A class is A but the return type of the get() method of B class is B. Both methods have different return type but it is method overriding. This is known as covariant return type.

**Q. What is final variable?**

**Ans. Final Keyword in Java**

The final keyword in java is used to restrict the user. The final keyword can be used in many context.

Final can be:

1. variable

2. method

3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.

**1) final variable**

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

**Example of final variable**

There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

class Bike{

final int speedlimit=90;//final variable

void run(){

speedlimit=400;

}

public static void main(String args[]){

Bike obj=new Bike();

obj.run();

}

Output:Compile Time Error

**Q. What is final method?**

**Ans.** If you make any method as final, you cannot override it.

Example of **final method**

class Bike{

final void run(){System.out.println("running");}

}

class Honda extends Bike{

void run(){System.out.println("running safely with 100kmph");}

public static void main(String args[]){

Honda honda= new Honda();

honda.run();

}

}

Output:Compile Time Error

**Q. What is final class?**

**Ans.** If you make any class as final, you cannot extend it.

Example of final class

final class Bike{}

class Honda extends Bike{

void run(){System.out.println("running safely with 100kmph");}

public static void main(String args[]){

Honda honda= new Honda();

honda.run();

}

Output:Compile Time Error

**Q. Is final method inherited?**

**Ans.** Yes, final method is inherited but you cannot override it. For Example:

class Bike{

final void run(){System.out.println("running...");}

}

class Honda extends Bike{

public static void main(String args[]){

new Honda().run(); }

}

Output:running…

**Q. What is blank final variable?**

**Ans.** A final variable that is not initialized at the time of declaration is known as blank final

variable. If you want to create a variable that is initialized at the time of creating object and once initialized may not be changed, it is useful. For example PAN CARD number of an employee. It can be initialized only in constructor.

Example of blank final variable

class Student{

int id;

String name;

final String PAN\_CARD\_NUMBER;

...

}

**Q. Can we initialize blank final variable?**

**Ans.** Yes, but only in constructor. For example:

class Bike{

final int speedlimit;//blank final variable

Bike(){

speedlimit=70;

System.out.println(speedlimit);

}

public Static void main(String args[]){

new Bike();

}

}

Output:70

**Q. What is final parameter?**

**Ans.** If you declare any parameter as final, you cannot change the value of it.

class Bike{

int cube(final int n){

n=n+2;//can't be changed as n is final

n\*n\*n;

}

public Static void main(String args[]){

Bike b=new Bike();

b.cube(5);

}

}

Output:Compile Time Error

**Q. Can we declare a constructor final?**

**Ans.** No, because constructor is never inherited.

**Q. Can you declare the main method as final?**

**Ans.** Yes, such as, public static final void main(String[] args) {}.

**Q. What is Runtime Polymorphism?**

**Ans. Runtime Polymorphism**

Runtime polymorphism or Dynamic Method Dispatch is a process in which a call to an overridden method is resolved at runtime rather than 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.

Let's first understand the upcasting before Runtime Polymorphism.

Upcasting

When reference variable of Parent class refers to the object of Child class, it is known as upcasting. For example:

class A{}

class B extends A{

A a=new B();//upcasting

}

**Q. Can you achieve Runtime Polymorphism by data members?**

**Ans.** Runtime Polymorphism with data member.Method is overriden not the data members, so runtime polymorphism can't be achieved by data members.

In the example given below, both the classes have a datamember speed limit, we are accessing the datamember by the reference variable of Parent class which refers to the subclass object. Since we are accessing the datamember which is not overridden, hence it will access the datamember of Parent class always.

Rule: Runtime polymorphism can't be achieved by data members.

**Q. What is the difference between static binding and dynamic binding?**

**Ans. Static Binding and Dynamic Binding**

Connecting a method call to the method body is known as binding. There are two types of binding

1. static binding (also known as early binding).

2. dynamic binding (also known as late binding).

Understanding Type

Let's understand the type of instance.

1) variables have a type

Each variable has a type, it may be primitive and non-primitive.

1. int data=30;

Here data variable is a type of int.

2) References have a type

1. class Dog{

2. public static void main(String args[]){

3. Dog d1;//Here d1 is a type of Dog

4. }

5. }

3) Objects have a type

An object is an instance of particular java class,but it is also an instance of its superclass.

1. class Animal{}

3. class Dog extends Animal{

4. public static void main(String args[]){

5. Dog d1=new Dog();

6. }

7. }

Here d1 is an instance of Dog class, but it is also an instance of Animal.

**static binding**

**When type of the object is determined at compile time(by the compiler), it is known as static binding.If there is any private, final or static method in a class, there is static binding.**

Example of static binding

1. class Dog{

2. private void eat(){System.out.println("dog is eating...");}

4. public static void main(String args[]){

5. Dog d1=new Dog();

6. d1.eat();

7. }

8. }

**Dynamic binding**

**When type of the object is determined at run-time, it is known as dynamic binding.**

Example of dynamic binding:

1. class Animal{

2. void eat(){System.out.println("animal is eating...");}

3. }

5. class Dog extends Animal{

6. void eat(){System.out.println("dog is eating...");}

8. public static void main(String args[]){

9. Animal a=new Dog();

10. a.eat();

11. }

12. }

Output:dog is eating...

In the above example object type cannot be determined by the compiler, because the instance of Dog is also an instance of Animal.So compiler doesn't know its type, only its base type.

**Q. What is abstraction?**

**Ans.** Abstraction is a process of hiding the implementation details and showing only functionality to the user. Another way, it shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery.

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

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)

2. Interface (100%)

**Q. What is abstract class?**

**Ans. Abstract class**

A class that is declared as abstract is known as abstract class. It needs to be extended and its method implemented. It cannot be instantiated.

Syntax to declare the abstract class

1. abstract class <class\_name>{}

**abstract method**

**A method that is declared as abstract and does not have implementation is known as abstract method.**

Syntax to define the abstract method

1. abstract return\_type <method\_name>();//no braces{}

Note: An abstract class can have data member, abstract method, method body, constructor and even main() method.

Rule: If there is any abstract method in a class, that class must be abstract.

Rule: If you are extending any abstract class that have abstract method, you must either provide the implementation of the method or make this class abstract.

**Q. What is marker interface?**

**Ans.** An interface that have no data member and method is known as a marker interface.For example Serializable,Cloneable etc.

**Nested Interface**

Note: An interface can have another interface i.e. known as nested interface. We will learn it in detail in the nested classes chapter. For example:

interface printable{

void print();

interface MessagePrintable{

void msg();

}

}

**Q. What is difference between abstract class and interface**

|  |  |
| --- | --- |
| **Abstract Class** | **Interface** |
| 1)An abstract class can have method body (non- abstract methods). | Interface have only abstract methods. |
| An abstract class can have instance variables. | An interface cannot have instance variables. |
| 3)An abstract class can have constructor. | Interface cannot have constructor. |
| 4)An abstract class can have static methods. | Interface cannot have static methods. |
| 5)You can extends one abstract class. | You can implement multiple interfaces. |

**Q. What is static import feature of Java5?**

**Ans.**

**Static Import:**

The static import feature of Java 5 facilitate the java programmer to access any static member of a class directly. There is no need to qualify it by the class name.

Advantage of static import:

● Less coding is required if you have access any static member of a class oftenly.

Disadvantage of static import:

● If you overuse the static import feature, it makes the program unreadable and unmaintainable.

**Q. What is the difference between import and static import?**

**Ans.** The import allows the java programmer to access classes of a package without package qualification whereas the static import feature allows to access the static members of a class without the class qualification. The import provides accessibility to classes and interface whereas static import provides accessibility to static members of the class.

**Q. What about package class?**

**Ans. Package class**

The package class provides methods to get information about the specification and implementation of a package. It provides methods such as getName(), getImplementationTitle(), getImplementationVendor(), getImplementationVersion() etc.

Example of Package class

In this example, we are printing the details of java.lang package by invoking the methods of package class.

class PackageInfo{

public static void main(String args[]){

Package p=Package.getPackage("java.lang");

System.out.println("package name: "+p.getName());

System.out.println("Specification Title: "+p.getSpecificationTitle());

System.out.println("Specification Vendor: "+p.getSpecificationVendor());

System.out.println("Specification Version: "+p.getSpecificationVersion());

System.out.println("Implementaion Title: "+p.getImplementationTitle());

System.out.println("Implementation Vendor: "+p.getImplementationVendor());

System.out.println("Implementation Version: "+p.getImplementationVersion());

System.out.println("Is sealed: "+p.isSealed());

}

}

**Output:**package name: java.lang

Specification Title: Java Plateform API Specification

Specification Vendor: Sun Microsystems, Inc.

Specification Version: 1.6

Implemenation Title: Java Runtime Environment

Implemenation Vendor: Sun Microsystems, Inc.

Implemenation Version: 1.6.0\_30

IS sealed: false

**Q. Can I import same package/class twice? Will the JVM load the package twice at runtime?**

**Ans.** One can import the same package or same class multiple times. Neither compiler nor JVM complains about it.But the JVM will internally load the class only once no matter how many times you import the same class.

**Q. How can we create immutable class in java ? How to create Immutable class?**

**Ans.** There are many immutable classes like String, Boolean, Byte, Short, Integer, Long, Float, Double etc. In short, all the wrapper classes and String class is immutable. We can also create immutable class by creating final class that have final data members as the example given below:

**Example to create Immutable class**

In this example, we have created a final class named Employee. It have one final datamember, a parameterized constructor and getter method.

public final class Employee{

final String pancardNumber;

public Employee(String pancardNumber){

this.pancardNumber=pancardNumber;

**Q. What is Garbage Collection?**

**Ans.**

**Garbage Collection:**

In java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically.

Advantage of Garbage Collection:

● It makes java memory efficient because garbage collector removes the unreferenced objects from heap memory.

● It is automatically done by the garbage collector so we don't need to make extra efforts.

How can an object be unreferenced?

There are many ways:

● By nulling the reference

● By assigning a reference to another

● By anonymous object etc.

**finalize() method:**

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in System class as:

1. protected void finalize(){}

Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).

**Q. What is gc()?**

**Ans.** gc() is a daemon thread.gc() method is defined in System class that is used to send request to JVM to perform garbage collection.

**Q. Can an unreferenced objects be referenced again?**

**Ans.** Yes

**Q. What kind of thread is the Garbage collector thread?**

**Ans.** Daemon thread.

**Q. What is difference between final, finally and finalize?**

**Ans.** final: final is a keyword, final can be variable, method or class.You, can't change the value of final variable, can't override final method, can't inherit final class.

finally: finally block is used in exception handling. finally block is always executed.

finalize():finalize() method is used in garbage collection.finalize() method is invoked just before the object is garbage collected.The finalize() method can be used to perform any cleanup processing.

**Q. What is the purpose of the Runtime class?**

**Ans.** The purpose of the Runtime class is to provide access to the Java runtime system.

**Q. How will you invoke any external process in Java?**

**Ans.** By Runtime.getRuntime().exec(?) method.

**Q. How do I convert a numeric IP address like 192.18.97.39 into a hostname like java.sun.com?**

**Ans.** By InetAddress.getByName("192.18.97.39").getHostName() where 192.18.97.39 is the IP address.

**Q. What is reflection?**

**Ans.** Reflection is the process of examining or modifying the runtime behaviour of a class at runtime.It is used in:

● IDE (Integrated Development Environment) e.g. Eclipse,MyEclipse,NetBeans.

● Debugger

● Test Tools etc.

**Q. Can you access the private method from outside the class?**

**Ans.** Yes, by changing the runtime behaviour of a class if the class is not secured.

**Q. What is a native method?**

**Ans.** A native method is a method that is implemented in a language other than Java.

**Q. What is the purpose of the System class?**

**Ans.** The purpose of the System class is to provide access to system resources.

**Q. What is an applet?**

**Ans.** An applet is a small java program that runs inside the browser and generates dynamic contents.

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

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

**Q. What is Locale?**

**Ans.** A Locale object represents a specific geographical, political, or cultural region.

**Q. How will you load a specific locale?**

**Ans.** By ResourceBundle.getBundle(?) method.

**Q. What is a JavaBean?**

**Ans.** Are reusable software components written in the Java programming language, designed to be manipulated visually by a software development environment, like JBuilder or VisualAge for Java.

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

**Ans.** 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).

**Q. What is Encapsulation?**

**Ans.** 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.

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

**Ans.**

* 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.

**Q. What is Polymorphism?**

**Ans.** 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.

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

**Ans.** (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).

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

**Ans.** 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

**Q. What is the difference between an Inner Class and a Sub-Class?**

**Ans.** An Inner class is a class which is nested within another class.

An Inner class has access rights for the class which is nesting it and it can access all variables and methods defined in the outer class.

A sub-class is a class which inherits from another class called superclass.

Subclass can access all public and protected methods and fields of its superclass.

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

**Ans.** 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.

**Q. What is Dynamic Binding?**

**Ans.** 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.

**Q. Can overloaded methods be overridden too?**

**Ans.** Yes, derived classes still can override the overloaded methods. Polymorphism can still happen. Compiler will not binding the method calls since it is overloaded, because it might be overridden now or in the future.

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

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

**Q.What is an Interface?**

**Ans.** 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.

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

**Ans.** 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.

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

**Ans.** 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.

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

**Ans.** 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 deserialized.

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

**Ans.** 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.

**Q.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. |

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

**Ans.** **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 non leaf classes in class hierarchies.

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

**Ans.** Yes, other non abstract methods can access a method that you declare as abstract.

**Q. What is Constructor?**

**Ans.**

* 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 can call the base class constructor.
* Constructor is invoked whenever an object of its associated class is created.

**Q. 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 |

**Q. 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. |

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

**Ans.** 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 |

**Q.What is final modifier?**

**Ans.** 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.

**Q. What are static methods?**

**Ans.** 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 orthis.