**1. What is the most important feature of Java?**

Java is a platform independent language.

**2. What do you mean by platform independence?**

Platform independence means that we can write and compile the java code in one platform (eg Windows) and can execute the class in any other supported platform eg (Linux,Solaris,etc)

**3. What is a JVM?**

JVM is Java Virtual Machine which is a run time environment for the compiled java class files.

**4. Are JVM's platform independent?**

JVM's are not platform independent. JVM's are platform specific run time implementation provided by the vendor.

**5. What is the difference between a JDK and a JVM?**

JDK is Java Development Kit which is for development purpose and it includes execution environment also. But JVM is purely a run time environment and hence you will not be able to compile your source files using a JVM.

**6. What is a pointer and does Java support pointers?**

Pointer is a reference handle to a memory location. Improper handling of pointers leads to memory leaks and reliability issues hence Java doesn't support the usage of pointers.

**7. What is the base class of all classes?**

java.lang.Object

**8. Does Java support multiple inheritance?**

Java doesn't support multiple inheritance.

**9. Is Java a pure object oriented language?**

Java uses primitive data types and hence is not a pure object oriented language.

**10. Are arrays primitive data types?**

In Java, Arrays are objects.

Private String [] s=new String[10];

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

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

**12. What are local variables?**

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

**13. What are instance variables?**

Instance variables are those which are defined at the class level. Instance variables need not be initialized before using them as they are automatically initialized to their default values.

**14. How to define a constant variable in Java?**

The variable should be declared as static and final. So only one copy of the variable exists for all instances of the class and the value can't be changed also.  
static final int PI = 2.14; is an example for constant.

**15. Should a main() method be compulsorily declared in all java classes?**

No not required. main() method should be defined only if the source class is a java application.

**16. What is the return type of the main() method?**

Main() method doesn't return anything hence declared void.

**17. Why is the main() method declared static?**

main() method is called by the JVM even before the instantiation of the class hence it is declared as static.

**18. What is the arguement of main() method?**

main() method accepts an array of String object as arguement.

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

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

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

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

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

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

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

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

**23. What is a package?**

Package is a collection of related classes and interfaces. package declaration should be first statement in a java class.

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

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

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

Not possible.

**26. Can a class be declared as protected?**

A class can't be declared as protected. only methods can be declared as protected.

**27. What is the access scope of a protected method?**

A protected method can be accessed by the classes within the same package or by the subclasses of the class in any package.

**28. What is the purpose of declaring a variable as final?**

A final variable's value can't be changed. final variables should be initialized before using them.

**29. What is the impact of declaring a method as final?**

A method declared as final can't be overridden. A sub-class can't have the same method signature with a different implementation.

**30. I don't want my class to be inherited by any other class. What should i do?**

You should declared your class as final. But you can't define your class as final, if it is an abstract class. A class declared as final can't be extended by any other class.

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

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

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

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.

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

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

**34. When will you define a method as static?**

When a method needs to be accessed even before the creation of the object of the class then we should declare the method as static.

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

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

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

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.

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

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.

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

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

**39. What is an Abstract Class and what is it's purpose?**

A Class which doesn't provide complete implementation is defined as an abstract class. Abstract classes enforce abstraction.

**40. Can a abstract class be declared final?**

Not possible. An abstract class without being inherited is of no use and hence will result in compile time error.

**41. What is use of a abstract variable?**

Variables can't be declared as abstract. only classes and methods can be declared as abstract.

**42. Can you create an object of an abstract class?**

Not possible. Abstract classes can't be instantiated.

**43. Can a abstract class be defined without any abstract methods?**

Yes it's possible. This is basically to avoid instance creation of the class.

**44. Class C implements Interface I containing method m1 and m2 declarations. Class C has provided implementation for method m2. Can i create an object of Class C?**

No not possible. Class C should provide implementation for all the methods in the Interface I. Since Class C didn't provide implementation for m1 method, it has to be declared as abstract. Abstract classes can't be instantiated.

**45. Can a method inside a Interface be declared as final?**

No not possible. Doing so will result in compilation error. public and abstract are the only applicable modifiers for method declaration in an interface.

**46. Can an Interface implement another Interface?**

Intefaces doesn't provide implementation hence a interface cannot implement another interface.

**47. Can an Interface extend another Interface?**

Yes an Interface can inherit another Interface, for that matter an Interface can extend more than one Interface.

**48. Can a Class extend more than one Class?**

Not possible. A Class can extend only one class but can implement any number of Interfaces.

**49. Why is an Interface be able to extend more than one Interface but a Class can't extend more than one Class?**

Basically Java doesn't allow multiple inheritance, so a Class is restricted to extend only one Class. But an Interface is a pure abstraction model and doesn't have inheritance hierarchy like classes(do remember that the base class of all classes is Object). So an Interface is allowed to extend more than one Interface.

**50. Can an Interface be final?**

Not possible. Doing so so will result in compilation error.

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

Yes it's possible.

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

Yes it's possible.

**53. What is a Marker Interface?**

An Interface which doesn't have any declaration inside but still enforces a mechanism.

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

Polymorphism.

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

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

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

No.

**57. What is Externalizable?**

Externalizable is an Interface that extends Serializable Interface. And sends data into Streams in Compressed Format. It has two methods, writeExternal(ObjectOuput out) and readExternal(ObjectInput in)

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

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

**59. What is a local, member and a class variable?**

Variables declared within a method are "local" variables.

Variables declared within the class i.e not within any methods are "member" variables (global variables).

Variables declared within the class i.e not within any methods and are defined as "static" are class variables.

**60. What is an abstract method?**

An abstract method is a method whose implementation is deferred to a subclass.

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

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

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

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

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

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.

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

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.

**65. What is the % operator?**

It is referred to as the modulo or remainder operator. It returns the remainder of dividing the first operand by the second operand.

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

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

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

The Object class is extended by all other classes.

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

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

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

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

**70. What is casting?**

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.

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

void.

**72. If a variable is declared as private, where may the variable be accessed?**

A private variable may only be accessed within the class in which it is declared.

**73. What do you understand by private, protected and public?**

These are accessibility modifiers. Private is the most restrictive, while public is the least restrictive. There is no real difference between protected and the default type (also known as package protected) within the context of the same package, however the protected keyword allows visibility to a derived class in a different package.

**74. What is Downcasting ?**

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

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

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

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

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.

**77. What restrictions are placed on the location of a package statement within a source code file?**

A package statement must appear as the first line in a source code file (excluding blank lines and comments).

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

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

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

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.

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

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

**81. What is the range of the char type?**

The range of the char type is 0 to 216 - 1 (i.e. 0 to 65535.)

**82. What is the range of the short type?**

The range of the short type is -(215) to 215 - 1. (i.e. -32,768 to 32,767)

**83. Why isn't there operator overloading?**

Because C++ has proven by example that operator overloading makes code almost impossible to maintain.

**84. What does it mean that a method or field is "static"?**

Static variables and methods are instantiated only once per class. In other words they are class variables, not instance variables. If you change the value of a static variable in a particular object, the value of that variable changes for all instances of that class. Static methods can be referenced with the name of the class rather than the name of a particular object of the class (though that works too). That's how library methods likeSystem.out.println() work. out is a static field in the java.lang.System class.

**85. Is null a keyword?**

The null value is not a keyword.

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

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.

**87. Is the ternary operator written x : y ? z or x ? y : z ?**

It is written x ?y : z.

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

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

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

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.

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

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

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

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

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

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

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

A while statement checks at the beginning of a loop to see whether the next loop iteration should occur. A do while statement checks at the end of a loop to see whether the next iteration of a loop should occur. The do while statement will always execute the body of a loop at least once.

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

A local inner class may be final or abstract.

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

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

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

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.

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

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

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

The values true and false are not keywords.

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

The result is a String object.

**100. What is the diffrence between inner class and nested class?**

When a class is defined within a scope od another class, then it becomes inner class. If the access modifier of the inner class is static, then it becomes nested class.

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

An abstract class may not be declared as final.

**102. What is numeric promotion?**

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.

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

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

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

The default value of the boolean type is false.

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

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.

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

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.

**107. What is a Java 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.

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

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

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

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.

**110. 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)?**

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 thejavax.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.HashMapinstead 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.

**111. Can a method be overloaded based on different return type but same argument type ?**

No, because the methods can be called without using their return type in which case there is ambiquity for the compiler.

**112. What happens to a static variable that is defined within a method of a class ?**

Can't do it. You'll get a compilation error.

**113. How many static initializers can you have ?**

As many as you want, but the static initializers and class variable initializers are executed in textual order and may not refer to class variables declared in the class whose declarations appear textually after the use, even though these class variables are in scope.

**114. What is the difference between method overriding and overloading?**

Overriding is a method with the same name and arguments as in a parent, whereas overloading is the same method name but different arguments

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

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.

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

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.

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

The = operator is right associative.

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

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

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

A break statement results in the termination of the statement to which it applies (switch, for, do, or while). Acontinue statement is used to end the current loop iteration and return control to the loop statement.

**120. Can a for statement loop indefinitely?**

Yes, a for statement can loop indefinitely. For example, consider the following: for(;;);

**121. To what value is a variable of the String type automatically initialized?**

The default value of an String type is null.

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

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.

**123. 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.

**124. What does it mean that a class or member is final?**

A final class cannot be inherited. A final method cannot be overridden in a subclass. A final field cannot be changed after it's initialized, and it must include an initializer statement where it's declared.

**125. What does it mean that a method or class is abstract?**

An abstract class cannot be instantiated. Abstract methods may only be included in abstract classes. However, an abstract class is not required to have any abstract methods, though most of them do. Each subclass of an abstract class must override the abstract methods of its superclasses or it also should be declared abstract.

**126. What is a transient variable?**

Transient variable is a variable that may not be serialized.

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

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

**128. What is the difference between the >> and >>> operators?**

The >> operator carries the sign bit when shifting right. The >>> zero-fills bits that have been shifted out.

**129. Is sizeof a keyword?**

The sizeof operator is not a keyword.

**1.what  is a transient variable?**

A transient variable is a variable that may not be serialized.

**2.which containers use a border Layout as their default layout?**

The window, Frame and Dialog classes use a border layout as their default layout.

**3.Why do threads block on I/O?**

Threads block on i/o (that is enters the waiting state) so that other threads may execute while the i/o Operation is performed.

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

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.

**5. What is synchronization and why is it important?**

With respect to multithreading, synchronization is the capability to control the access of multiple threads to shared resources. Without synchronization, it is possible for one thread to modify a shared object while another thread is in the process of using or updating that object's value. This often leads to significant errors.

**6. Can a lock be acquired on a class?**

Yes, a lock can be acquired on a class. This lock is acquired on the class's Class object.

**7. What's new with the stop(), suspend() and resume() methods in JDK 1.2?**

The stop(), suspend() and resume() methods have been deprecated in JDK 1.2.

**8. Is null a keyword?**

The null value is not a keyword.

**9. What is the preferred size of a component?**

The preferred size of a component is the minimum component size that will allow the component to display normally.

**10. What method is used to specify a container's layout?**

The setLayout() method is used to specify a container's layout.

**11. Which containers use a FlowLayout as their default layout?**

The Panel and Applet classes use the FlowLayout as their default layout.

**12. What state does a thread enter when it terminates its processing?**

When a thread terminates its processing, it enters the dead state.

**13. What is the Collections API?**

The Collections API is a set of classes and interfaces that support operations on collections of objects.

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

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.

**15. What is the List interface?**

The List interface provides support for ordered collections of objects.

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

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

**17. What is the Vector class?**

The Vector class provides the capability to implement a growable array of objects

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

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

**19. What is an Iterator interface?**

The Iterator interface is used to step through the elements of a Collection.

**20. What is the difference between the >> and >>> operators?**

The >> operator carries the sign bit when shifting right. The >>> zero-fills bits that have been shifted out.

**21. Which method of the Component class is used to set the position and size of a component?**

setBounds()

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

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.

**23What is the difference between yielding and sleeping?**

When a task invokes its yield() method, it returns to the ready state. When a task invokes its sleep() method, it returns to the waiting state.

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

The EventObject class and the EventListener interface support event processing.

**25. Is sizeof a keyword?**

The sizeof operator is not a keyword.

**26. What are wrapped classes?**

Wrapped classes are classes that allow primitive types to be accessed as objects.

**27. 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

**28. What restrictions are placed on the location of a package statement within a source code file?**

A package statement must appear as the first line in a source code file (excluding blank lines and comments).

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

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.

**30. What is the immediate superclass of the Applet class?**

Panel

**31. What is the difference between preemptive scheduling and time slicing?**

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.

**32. Name three Component subclasses that support painting.**

The Canvas, Frame, Panel, and Applet classes support painting.

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

The readLine() method returns null when it has reached the end of a file.

**34. What is the immediate superclass of the Dialog class?**

Window

**35. What is clipping?**

Clipping is the process of confining paint operations to a limited area or shape.

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

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

**37. Can a for statement loop indefinitely?**

Yes, a for statement can loop indefinitely. For example, consider the following: for(;;) ;

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

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

**39. When a thread blocks on I/O, what state does it enter?**

A thread enters the waiting state when it blocks on I/O.

**40. To what value is a variable of the String type automatically initialized?**

The default value of an String type is null.

**41. What is the catch or declare rule for method declarations?**

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.

**42. What is the difference between a MenuItem and a CheckboxMenuItem?**

The CheckboxMenuItem class extends the MenuItem class to support a menu item that may be checked or unchecked.

**43. What is a task's priority and how is it used in scheduling?**

A task's priority is an integer value 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.

**44. What class is the top of the AWT event hierarchy?**

The java.awt.AWTEvent class is the highest-level class in the AWT event-**class hierarchy.**

**45. When a thread is created and started, what is its initial state?**

A thread is in the ready state after it has been created and started.

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

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

**47. What is the range of the short type?**

The range of the short type is -(2^15) to 2^15 - 1.

**48. What is the range of the char type?**

The range of the char type is 0 to 2^16 - 1.

**49. In which package are most of the AWT events that support the event-delegation model defined?**

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.

**50. What is the immediate superclass of Menu?**

MenuItem

**51. 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.

**52. Which class is the immediate superclass of the MenuComponent class.**

Object

**53. What invokes a thread's run() method?**

After a thread is started, via its start() method or that of the Thread class, the JVM invokes the thread's run() method when the thread is initially executed.

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

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.

**55. Name three subclasses of the Component class.**

Box.Filler, Button, Canvas, Checkbox, Choice, Container, Label, List, Scrollbar, or TextComponent

**56. What is the GregorianCalendar class?**

The GregorianCalendar provides support for traditional Western calendars.

**57. Which Container method is used to cause a container to be laid out and redisplayed?**

validate()

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

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

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

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

**60. What is the purpose of the finally clause of a try-catch-finally statement?**

The finally clause is used to provide the capability to execute code no matter whether or not an exception is thrown or caught.

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

A program's main() method takes an argument of the String[] type.

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

The = operator is right associative.

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

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

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

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

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

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.

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

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

**67. What method is invoked to cause an object to begin executing as a separate thread?**

The start() method of the Thread class is invoked to cause an object to begin executing as a separate thread.

**68. Name two subclasses of the TextComponent class.**

TextField and TextArea

**69. What is the advantage of the event-delegation model over the earlier event-inheritance model?**

The event-delegation model has two advantages over the event-inheritance model. First, it enables event handling 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 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.

**70. Which containers may have a MenuBar?**

Frame

**71. How are commas used in the intialization and iteration  parts of a for statement?**

Commas are used to separate multiple statements within the initialization and iteration parts of a for statement.

**72. What is the purpose of the wait(), notify(), and notifyAll() methods?**

The wait(),notify(), and notifyAll() methods are used to provide an efficient way for threads to wait for a shared resource. When a thread executes an object's wait() method, it enters the waiting state. It only enters the ready state after another thread invokes the object's notify() or notifyAll() methods.

**73. What is an abstract method?**

An abstract method is a method whose implementation is deferred to a subclass.

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

A Java source code file takes the name of a public class or interface that is defined within the file. A source code 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 extension.

**75. What is the relationship between the Canvas class and the Graphics class?**

A Canvas object provides access to a Graphics object via its paint() method.

**76. What are the high-level thread states?**

The high-level thread states are ready, running, waiting, and dead.

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

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

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

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

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

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.

**80. What is the difference between the String and StringBuffer classes?**

String objects are constants. StringBuffer objects are not.

**81. If a variable is declared as private, where may the variable be accessed?**

A private variable may only be accessed within the class in which it is declared.

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

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.

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

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

**84. How are the elements of a BorderLayout organized?**

The elements of a BorderLayout are organized at the borders (North, South, East, and West) and the center of a container.

**85. What is the % operator?**

It is referred to as the modulo or remainder operator. It returns the remainder of dividing the first operand by the second operand.

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

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

**87. What is the difference between a Window and a Frame?**

The Frame class extends Window to define a main application window that can have a menu bar.

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

The Object class is extended by all other classes.

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

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

**90. Is the ternary operator written x : y ? z or x ? y : z ?**

It is written x ? y : z.

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

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

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

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

**93. What happens when a thread cannot acquire a lock on an object?**

If a thread attempts to execute a synchronized method or synchronized statement and is unable to acquire an object's lock, it enters the waiting state until the lock becomes available.

**94. What is the difference between the Reader/Writer class hierarchy and the InputStream/OutputStream class hierarchy?**

The Reader/Writer class hierarchy is character-oriented, and the InputStream/OutputStream class hierarchy is byte-oriented.

**95. What classes of exceptions may be caught by a catch clause?**

A catch clause can catch any exception that may be assigned to the Throwable type. This includes the Error and Exception types.

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

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.

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

The SimpleTimeZone class provides support for a Gregorian calendar.

**98. What is the Map interface?**

The Map interface replaces the JDK 1.1 Dictionary class and is used associate keys with values.

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

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

**100. For which statements does it make sense to use a label?**

The only statements for which it makes sense to use a label are those statements that can enclose a break or continue statement.

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

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

**102. Which TextComponent method is used to set a TextComponent to the read-only state?**

setEditable()

**103. How are the elements of a CardLayout organized?**

The elements of a CardLayout are stacked, one on top of the other, like a deck of cards.

**104. Is &&= a valid Java operator?**

No, it is not.

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

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

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

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

**107. What is the relationship between clipping and repainting?**

When a window is repainted by the AWT painting thread, it sets the clipping regions to the area of the window that requires repainting.

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

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

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

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.

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

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

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

An interface may be declared as public or abstract.

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

A class is a subclass of itself.

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

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

**114. What event results from the clicking of a button?**

The ActionEvent event is generated as the result of the clicking of a button.

**115. How can a GUI component handle its own events?**

A component can handle its own events by implementing the required event-listener interface and adding itself as its own event listener.

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

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.

**117. How are the elements of a GridBagLayout organized?**

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.

**118. What advantage do Java's layout managers provide over traditional windowing systems?**

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.

**119. What is the Collection interface?**

The Collection interface provides support for the implementation of a mathematical bag - an unordered collection of objects that may contain duplicates.

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

A local inner class may be final or abstract.

**121. 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.

**122. What is the difference between the paint() and repaint() methods?**

The paint() method supports painting via a Graphics object. The repaint() method is used to cause paint() to be invoked by the AWT painting thread.

**123. What is the purpose of the File class?**

The File class is used to create objects that provide access to the files and directories of a local file system.

**124. Can an exception be rethrown?**

Yes, an exception can be rethrown.

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

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

126. How does multithreading take place on a computer with a single CPU?

The operating system's task scheduler allocates execution time to multiple tasks. By quickly switching between executing tasks, it creates the impression that tasks execute sequentially.

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

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

**128. When is the finally clause of a try-catch-finally statement executed?**

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.

**129. Which class is the immediate superclass of the Container class?**

Component

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

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.

**131. How can the Checkbox class be used to create a radio button?**

By associating Checkbox objects with a CheckboxGroup.

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

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

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

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

**134. What happens when you invoke a thread's interrupt method while it is sleeping or waiting?**

When a task's interrupt() method is executed, the task enters the ready state. The next time the task enters the running state, an InterruptedException is thrown.

**135. What is casting?**

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.

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

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

**137. Name four Container classes.**

Window, Frame, Dialog, FileDialog, Panel, Applet, or ScrollPane

**138. What is the difference between a Choice and a List?**

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 are visible. A List supports the selection of one or more List items.

**139. What class of exceptions are generated by the Java run-time system?**

The Java runtime system generates RuntimeException and Error exceptions.

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

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

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

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.

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

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

**143. 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.

**144. What is the relationship between a method's throws clause and the exceptions that can be thrown during the method's execution?**

A method's throws clause must declare any checked exceptions that are not caught within the body of the method.

**145. What is the difference between the JDK 1.02 event model and the event-delegation model introduced with JDK 1.1?**

The JDK 1.02 event model 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 unhandled events.

**146. How is it possible for two String objects with identical values not to be equal under the == operator?**

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.

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

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

**148. What Checkbox method allows you to tell if a Checkbox is checked?**

getState()

**149. What state is a thread in when it is executing?**

An executing thread is in the running state.

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

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

**151. How are the elements of a GridLayout organized?**

The elements of a GridBad layout are of equal size and are laid out using the squares of a grid.

**152. What an I/O filter?**

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.

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

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

**154. What is the Set interface?**

The Set interface provides methods for accessing the elements of a finite mathematical set. Sets do not allow duplicate elements.

**155. What classes of exceptions may be thrown by a throw statement?**

A throw statement may throw any expression that may be assigned to the Throwable type.

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

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

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

The values true and false are not keywords.

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

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

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

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.

**160. What is the difference between the File and RandomAccessFile classes?**

The File class encapsulates the files and directories of the local file system. The RandomAccessFile class provides the methods needed to directly access data contained in any part of a file.

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

The result is a String object.

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

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..

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

The java.lang package is always imported by default.

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

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

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

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.

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

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.

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

A compilation unit is a Java source code file.

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

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

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

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.

**170. How can a dead thread be restarted?**

A dead thread cannot be restarted.

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

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.

**172. What is a layout manager?**

A layout manager is an object that is used to organize components in a container.

**173. Which arithmetic operations can result in the throwing of an ArithmeticException?**

Integer / and % can result in the throwing of an ArithmeticException.

**174. What are three ways in which a thread can enter the waiting state?**

A thread can enter the waiting state by invoking its sleep() method, by blocking on I/O, by unsuccessfully attempting to acquire an object's lock, or by invoking an object's wait() method. It can also enter the waiting state by invoking its (deprecated) suspend() method.

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

An abstract class may not be declared as final.

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

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.

**177. 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?**

The exception propagates up to the next higher level try-catch statement (if any) or results in the program's termination.

**178. What is numeric promotion?**

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.

**179. What is the difference between a Scrollbar and a ScrollPane?**

A Scrollbar is a Component, but not a Container. A ScrollPane is a Container. A ScrollPane handles its own events and performs its own scrolling.

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

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

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

The default value of the boolean type is false.

**182. Can try statements be nested?**

Try statements may be tested.

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

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.

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

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

**185. What is a Java 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.

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

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

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

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.

**188. 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.

**189. Can an unreachable object become reachable again?**

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.

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

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

**191. What method must be implemented by all threads?**

All tasks must implement the run() method, whether they are a subclass of  Thread or implement the Runnable interface.

**192. What methods are used to get and set the text label displayed by a Button object?**

getLabel() and setLabel()

**193. Which Component subclass is used for drawing and painting?**

Canvas

**194. What are synchronized methods and synchronized statements?**

Synchronized methods are methods that are used to control access to an object. A thread only executes a synchronized method after it has acquired the lock for the method's object or class. Synchronized statements are similar to synchronized methods. A synchronized statement can only be executed after a thread has acquired the lock for the object or class referenced in the synchronized statement.

**195. What are the two basic ways in which classes that can be run as threads may be defined?**

A thread class may be declared as a subclass of Thread, or it may implement the Runnable interface.

**196. What are the problems faced by Java programmers who don't use layout managers?**

Without layout managers, Java programmers are faced with determining how their GUI will be displayed across multiple windowing systems and finding a common sizing  and positioning that will work within the constraints imposed by each windowing system.

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

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.

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

The result is a String object.

**199. What is the List interface?**

The List interface provides support for ordered collections of objects.

**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: Name the containers which uses Border Layout as their default layout?  
Answer:** Containers which uses Border Layout as their default are: window, Frame and Dialog classes.

**Question: What do you understand by Synchronization?  
Answer:** 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.  
   }  
}

**Question: What is Collection API?  
Answer:** The Collection API is a set of classes and interfaces that support operation on collections of objects. These classes and interfaces are more flexible, more powerful, and more regular than the vectors, arrays, and hashtables if effectively replaces.   
**Example of classes**: HashSet, HashMap, ArrayList, LinkedList, TreeSet and TreeMap.  
**Example of interfaces**: Collection, Set, List and Map.

**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 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;   
}  
 **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.
* 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, 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.

**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  
}  
}

**1.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 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 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 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.Serializableinterface 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 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 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.

**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.*

**46.What is an Iterator ?**

* The Iterator interface is used to step through the elements of a Collection.
* Iterators let you process each element of a Collection.
* Iterators are a generic way to go through all the elements of a Collection no matter how it is organized.
* Iterator is an Interface implemented a different way for every Collection.

**47.How do you traverse through a collection using its Iterator?**

To use an iterator to traverse through the contents of a collection, follow these steps:

* Obtain an iterator to the start of the collection by calling the collections ***iterator()*** method.
* Set up a loop that makes a call to ***hasNext()***. Have the loop iterate as long as ***hasNext()*** returns **true**.
* Within the loop, obtain each element by calling **next()**.

**48.How do you remove elements during Iteration?**

Iterator also has a method ***remove()*** when remove is called, the current element in the iteration is deleted.

**49.What is the difference between Enumeration and Iterator?**

|  |  |
| --- | --- |
| **Enumeration** | **Iterator** |
| Enumeration doesn't have a remove() method | Iterator has a remove() method |
| Enumeration acts as Read-only interface, because it has the methods only to traverse and fetch the objects | Can be *abstract, final, native, static*, or *synchronized* |

**Note**: So Enumeration is used whenever we want to make Collection objects as Read-only.

|  |
| --- |
|  |

**50.How is ListIterator?**

**ListIterator** is just like Iterator, except it allows us to access the collection in either the forward or backward direction and lets us modify an element

**51.What is the List interface?**

* The List interface provides support for ordered collections of objects.
* Lists may contain duplicate elements.

**52.What are the main implementations of the List interface ?**

The main implementations of the List interface are as follows :

* **ArrayList** : Resizable-array implementation of the List interface. The best all-around implementation of the List interface.
* **Vector** : Synchronized resizable-array implementation of the List interface with additional "legacy methods."
* **LinkedList** : Doubly-linked list implementation of the List interface. May provide better performance than the ArrayList implementation if elements are frequently inserted or deleted within the list. Useful for queues and double-ended queues (deques).

**53.What are the advantages of ArrayList over arrays ?**

Some of the advantages ArrayList has over arrays are:

* It can grow dynamically
* It provides more powerful insertion and search mechanisms than arrays.

**54.Difference between ArrayList and Vector ?**

|  |  |
| --- | --- |
| **ArrayList** | **Vector** |
| ArrayList is **NOT** synchronized by default. | Vector List is synchronized by default. |
| ArrayList can use only Iterator to access the elements. | Vector list can use Iterator and Enumeration Interface to  access the elements. |
| The ArrayList increases its array size by 50 percent if it runs out of room. | A Vector defaults to doubling the size of its array if it runs  out of room |
| ArrayList has no default size. | While vector has a default size of 10. |

**55.How to obtain Array from an ArrayList ?**

Array can be obtained from an ArrayList using ***toArray()***method on ArrayList.

List arrayList = new ArrayList();  
 arrayList.add(â€¦

ObjectÂ  a[] = **arrayList.toArray()**;

**56.Why insertion and deletion in ArrayList is slow compared to LinkedList ?**

* **ArrayList**internally uses and array to store the elements, when that array gets filled by inserting elements a new array of roughly 1.5 times the size of the original array is created and all the data of old array is copied to new array.
* During deletion, all elements present in the array after the deleted elements have to be moved one step back to fill the space created by deletion. In linked list data is stored in nodes that have reference to the previous node and the next node so adding element is simple as creating the node an updating the next pointer on the last node and the previous pointer on the new node. Deletion in linked list is fast because it involves only updating the next pointer in the node before the deleted node and updating the previous pointer in the node after the deleted node.

|  |
| --- |
|  |

**57.Why are Iterators returned by ArrayList called Fail Fast ?**

Because, if list is structurally modified at any time after the iterator is created, in any way except through the iterator's own remove or add methods, the iterator will throw a ConcurrentModificationException. Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

**58.How do you decide when to use ArrayList and When to use LinkedList?**

If you need to support random access, without inserting or removing elements from any place other than the end, then ArrayList offers the optimal collection. If, however, you need to frequently add and remove elements from the middle of the list and only access the list elements sequentially, then LinkedList offers the better implementation.

**59.What is the Set interface ?**

* The Set interface provides methods for accessing the elements of a finite mathematical set
* Sets do not allow duplicate elements
* Contains no methods other than those inherited from Collection
* It adds the restriction that duplicate elements are prohibited
* Two Set objects are equal if they contain the same elements

**60.What are the main Implementations of the Set interface ?**

The main implementations of the List interface are as follows:

* HashSet
* TreeSet
* LinkedHashSet
* EnumSet

**61.What is a HashSet ?**

* A HashSet is an unsorted, unordered Set.
* It uses the hashcode of the object being inserted (so the more efficient your hashcode() implementation the better access performance you’ll get).
* Use this class when you want a collection with no duplicates and you don’t care about order when you iterate through it.

**62.What is a TreeSet ?**

TreeSet is a Set implementation that keeps the elements in sorted order. The elements are sorted according to the natural order of elements or by the comparator provided at creation time.

**63.What is an EnumSet ?**

An EnumSet is a specialized set for use with enum types, all of the elements in the EnumSet type that is specified, explicitly or implicitly, when the set is created.

**64.Difference between HashSet and TreeSet ?**

|  |  |
| --- | --- |
| **HashSet** | **TreeSet** |
| HashSet is under set interface i.e. it  does not guarantee for either sorted order or sequence order. | TreeSet is under set i.e. it provides elements  in a sorted  order (acceding order). |
| We can add any type of elements to hash set. | We can add only similar types  of elements to tree set. |

|  |
| --- |
|  |

**65.What is a Map ?**

* A map is an object that stores associations between keys and values (key/value pairs).
* Given a key, you can find its value. Both keys  and  values are objects.
* The keys must be unique, but the values may be duplicated.
* Some maps can accept a null key and null values, others cannot.

**66.What are the main Implementations of the Map interface ?**

The main implementations of the List interface are as follows:

* HashMap
* HashTable
* TreeMap
* EnumMap

**67.What is a TreeMap ?**

TreeMap actually implements the SortedMap interface which extends the Map interface. In a TreeMap the data will be sorted in ascending order of keys according to the natural order for the key's class, or by the comparator provided at creation time. TreeMap is based on the Red-Black tree data structure.

**68.How do you decide when to use HashMap and when to use TreeMap ?**

For inserting, deleting, and locating elements in a Map, the HashMap offers the best alternative. If, however, you need to traverse the keys in a sorted order, then TreeMap is your better alternative. Depending upon the size of your collection, it may be faster to add elements to a HashMap, then convert the map to a TreeMap for sorted key traversal.

**69.Difference between HashMap and Hashtable ?**

|  |  |
| --- | --- |
| **HashMap** | **Hashtable** |
| HashMap lets you have null values as well as one null key. | HashTable  does not allows null values as key and value. |
| The iterator in the HashMap is fail-safe (If you change the map while iterating, you’ll know). | The enumerator for the Hashtable is not fail-safe. |
| HashMap is unsynchronized. | Hashtable is synchronized. |

**Note**: Only one NULL is allowed as a key in HashMap. HashMap does not allow multiple keys to be NULL. Nevertheless, it can have multiple NULL values.

**70.How does a Hashtable internally maintain the key-value pairs?**

TreeMap actually implements the SortedMap interface which extends the Map interface. In a TreeMap the data will be sorted in ascending order of keys according to the natural order for the key's class, or by the comparator provided at creation time. TreeMap is based on the Red-Black tree data structure.

|  |
| --- |
|  |

**71.What Are the different Collection Views That Maps Provide?**

Maps Provide Three Collection Views.

* **Key Set**- allow a map's contents to be viewed as a set of keys.
* **Values Collection** - allow a map's contents to be viewed as a set of values.
* **Entry Set** - allow a map's contents to be viewed as a set of key-value mappings.

**72.What is a KeySet View ?**

KeySet is a set returned by the ***keySet()*** method of the Map Interface, It is a set that contains all the keys present in the Map.

**73.What is a Values Collection View ?**

Values Collection View is a collection returned by the ***values()*** method of the Map Interface, It contains all the objects present as values in the map.

**74.What is an EntrySet View ?**

Entry Set view is a set that is returned by the ***entrySet()*** method in the map and contains Objects of type Map. Entry each of which has both Key and Value.

**75.How do you sort an ArrayList (or any list) of user-defined objects ?**

Create an implementation of the *java.lang.Comparable* interface that knows how to order your objects and pass it to *java.util.Collections.sort*(List, Comparator).

**76.What is the Comparable interface ?**

The Comparable interface is used to sort collections and arrays of objects using the Collections.sort() and java.utils.Arrays.sort() methods respectively. The objects of the class implementing the Comparable interface can be ordered.

The Comparable interface in the generic form is written as follows:

interface Comparable<T>

*where T is the name of the type parameter.*  
  
All classes implementing the Comparable interface must implement the compareTo() method that has the return type as an integer. The signature of thecompareTo() method is as follows:

int i = object1.compareTo(object2)

* If object1 < object2: The value of i returned will be negative.
* If object1 > object2: The value of i returned will be positive.
* If object1 = object2: The value of i returned will be zero.

**77.What are the differences between the Comparable and Comparator interfaces ?**

|  |  |
| --- | --- |
| **Comparable** | **Comparator** |
| It uses the *compareTo()* method.  *int objectOne.compareTo(objectTwo).* | It uses the *compare()*method.  *int compare(ObjOne, ObjTwo)* |
| It is necessary to modify the class whose instance is going to be sorted. | A separate class can be created in order to sort the instances. |
| Only one sort sequence can be created. | Many sort sequences can be created. |
| It is frequently used by the API classes. | It used by third-party classes to sort instances. |

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

The class System 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);

**2. Question: How you can force the garbage collection?**

Garbage collection automatic process and can’t be forced. You could request it by calling System.gc(). JVM does not guarantee that GC will be started immediately.

*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 can’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.

**3. What’s the difference between constructors and normal methods?**

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 and it can return a value or can be void.

**4.Explain the usage of Java packages.**

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

**5.What is Collection API?**

The Collection API is a set of classes and interfaces that support operation on collections of objects. These classes and interfaces are more flexible, more powerful, and more regular than the vectors, arrays, and hashtables if effectively replaces.  
Example of classes: HashSet, HashMap, ArrayList, LinkedList, TreeSet and TreeMap.  
Example of interfaces: Collection, Set, List and Map.

**6.Explain the usage of the keyword transient?**

Transient 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 of its data type (i.e. zero for integers).

**7.What’s the difference between the methods sleep() and wait()**

The code sleep(1000); puts thread aside for exactly one second. The code wait(1000), causes a wait of up to one second. A thread could stop waiting earlier if it receives the notify() or notifyAll() call. The method wait() is defined in the class Object and the method sleep() is defined in the class Thread.

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

Synchronized blocks place locks for shorter periods than synchronized methods.

**9.Can an inner class declared inside of a method access local variables of this method?**It’s possible if these variables are final.

**10.Explain the user defined Exceptions?**

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  
}

**11.Describe the wrapper classes in Java.**

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

**12.Which of the following are valid definitions of an application’s main( ) method?**

a) public static void main();  
b) public static void main( String args );  
c) public static void main( String args[] );  
d) public static void main( Graphics g );  
e) public static boolean main( String args[] );

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

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.  
  
**13. Does all property of immutable object needs to be final?**  
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.  
  
**14. What is the difference between creating String as new () and literal?**  
When we create string with new () it’s created in heap and not added into string pool while String created using literal are created in String pool itself which exists in Perm area of heap.

String s = new String("Test");  
will put the object in String pool , it it does then why do one need String.intern() method which is used to put Strings into String pool explicitly. its only when you create String object as String literal e.g. String s = "Test" it put the String in pool.  
  
15. How does substring () inside String works?  
Another good question, I think answer is not sufficient but here it is “Substring creates new object out of string by taking a portion of original string”.  
  
  
The substring() method is used to return a part (or substring) of the String used to invoke the method. The first argument represents the starting location (zero-based) of the substring. If the call has only one argument, the substring returned will include the characters to the end of the original String. If the call has two arguments, the substring returned will end with the character located in the nth position of the original String where n is the second argument. Unfortunately, the ending argument is not zero-based, so if the second argument is 7, the last character in the returned String will be in the original String’s 7 position, which is index 6. Let’s look at an example:  
String x = "0123456789";  
  
System.out.println( x.substring(5) ); // output is "56789"  
System.out.println( x.substring(5, 8)); // output is "567"  
  
The first example should be easy: start at index 5 and return the rest of the String. The second example should be read as follows: start at index 5 and return the characters up to and including the 8th position (index 7).  
  
  
and @Anonymous pointed out some interesting fact:  
omething important about String.substring() method, its implementation calls the following String(...) constructor :  
  
// Package private constructor which shares value array for speed.  
String(int offset, int count, char value[]) {  
this.value = value;  
this.offset = offset;  
this.count = count;  
}  
  
That means the new String() object returned by substring(...) shares the same backing array (this.value) as the original string object.  
  
Thus if your original string object is 1GB long, the substring object will always be 1GB long too!  
  
You will probably want to trim the new array size to the substring range, this should do the job:  
  
String veryLongString = readFromFile(file);  
String tinyString = new String(veryLongString.substring(1,10));  
  
The String(String) constructor is implemented that way:  
  
public String(String original) {  
...  
if (originalValue.length > size) {  
// The array representing the String is bigger than the new  
// String itself. Perhaps this constructor is being called  
// in order to trim the baggage, so make a copy of the array.  
int off = original.offset;  
v = Arrays.copyOfRange(originalValue, off, off+size);  
}  
...  
}  
  
  
**16. Which two method you need to implement for key in hashMap ?**  
(equals and hashcode) read [How HashMap works in Java](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html) for detailed explanation.  
  
**17. Where does these  two method comes in picture during get operation?**  
See here  [How HashMap works in Java](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html) for detailed explanation.  
  
**18. How do you handle error condition while writing stored procedure or accessing stored procedure from java?**  
Open for all, my friend didn't know the answer so he didn't mind telling me.  
  
**19. What is difference between Executor.submit() and Executer.execute() method ?**  
  
There is a difference when looking at exception handling. If your tasks throws anexception and if it was submitted with execute this exception will go to the uncaught exception handler (when you don't have provided one explicitly, the default one will just print the stack trace to System.err). If you submitted the task with submit any thrown exception, checked or not, is then part of the task's return status. For a task that was submitted with submit and that terminates with anexception, the Future.get will rethrow this exception, wrapped in an ExecutionException.  
  
**20. When do you override hashcode and equals() ?**  
Whenever necessary especially if you want to do equality check or want to use your object as key in HashMap. check this for writing equals method correctly    
  
**21. What will be the problem if you don't override hashcode() method ?**  
You will not be able to recover your object from hash Map if that is used as key in HashMap.   
See here  [How HashMap works in Java](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html) for detailed explanation.  
  
  
  
**22. What is the difference when String is gets created using literal or new() operator ?**  
When we create string with new() its created in heap and not added into string pool while String created using literal are created in String pool itself which exists in Perm area of heap.  
  
**23. Does not overriding hashcode() method has any performance implication ?**  
This is a good question and open to all , as per my knowledge a poor hashcode function will result in frequent collision in HashMap which eventually increase time for adding an object into Hash Map.  
  
**24. Give a simplest way to find out the time a method takes for execution without using any** profiling tool?  
Read the system time just before the method is invoked and immediately after method returns. Take the time difference, which will give you the time taken by a method for execution.  
  
To put it in code…  
  
long start = System.currentTimeMillis ();  
method ();  
long end = System.currentTimeMillis ();  
  
System.out.println (“Time taken for execution is ” + (end – start));  
  
Remember that if the time taken for execution is too small, it might show that it is taking zero milliseconds for execution. Try it on a method which is big enough, in the sense the one which is doing considerable amout of processing

**COLLECTION AND COLLECCTION FRAMEWORKS**

1. **[What is Java Collections API?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "Java-Collections-API)**

Java Collections framework API is a unified architecture for representing and manipulating collections. The API contains Interfaces, Implementations & Algorithm to help java programmer in everyday programming. In nutshell, this API does 6 things at high level

* + Reduces programming efforts. - Increases program speed and quality.
  + Allows interoperability among unrelated APIs.
  + Reduces effort to learn and to use new APIs.
  + Reduces effort to design new APIs.
  + Encourages & Fosters software reuse.

To be specific, There are six collection java interfaces. The most basic interface is Collection. Three interfaces extend Collection: Set, List, and SortedSet. The other two collection interfaces, Map and SortedMap, do not extend Collection, as they represent mappings rather than true collections.

1. **[What is an Iterator?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "What-is-Iterator)**

Some of the collection classes provide traversal of their contents via a java.util.Iterator interface. This interface allows you to walk through a collection of objects, operating on each object in turn. Remember when using Iterators that they contain a snapshot of the collection at the time the Iterator was obtained; generally it is not advisable to modify the collection itself while traversing an Iterator.

1. **[What is the difference between java.util.Iterator and java.util.ListIterator?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "Iterator-vs-ListIterator)**

Iterator : Enables you to traverse through a collection in the forward direction only, for obtaining or removing elements ListIterator : extends Iterator, and allows bidirectional traversal of list and also allows the modification of elements.

1. **[What is HashMap and Map?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "What-is-HashMap)**

Map is Interface which is part of Java collections framework. This is to store Key Value pair, and Hashmap is class that implements that using hashing technique.

1. **[Difference between HashMap and HashTable? Compare Hashtable vs HashMap?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "HashMap-vs-HashTable)**

Both Hashtable & HashMap provide key-value access to data. The Hashtable is one of the original collection classes in Java (also called as legacy classes). HashMap is part of the new Collections Framework, added with Java 2, v1.2. There are several differences between HashMap and Hashtable in Java as listed below

* + The HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls. (HashMap allows null values as key and value whereas Hashtable doesn’t allow nulls).
  + HashMap does not guarantee that the order of the map will remain constant over time. But one of HashMap's subclasses is LinkedHashMap, so in the event that you'd want predictable iteration order (which is insertion order by default), you can easily swap out the HashMap for a LinkedHashMap. This wouldn't be as easy if you were using Hashtable.
  + HashMap is non synchronized whereas Hashtable is synchronized.
  + Iterator in the HashMap is fail-fast while the enumerator for the Hashtable isn't. So this could be a design consideration.

1. **[What does synchronized means in Hashtable context?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "synchronized-Hashtable)**

Synchronized means only one thread can modify a hash table at one point of time. Any thread before performing an update on a hashtable will have to acquire a lock on the object while others will wait for lock to be released.

1. **[What is fail-fast property?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "fail-fast-property)**

At high level - Fail-fast is a property of a system or software with respect to its response to failures. A fail-fast system is designed to immediately report any failure or condition that is likely to lead to failure. Fail-fast systems are usually designed to stop normal operation rather than attempt to continue a possibly-flawed process.   
  
When a problem occurs, a fail-fast system fails immediately and visibly. Failing fast is a non-intuitive technique: "failing immediately and visibly" sounds like it would make your software more fragile, but it actually makes it more robust. Bugs are easier to find and fix, so fewer go into production.   
  
In Java, Fail-fast term can be related to context of iterators. If an iterator has been created on a collection object and some other thread tries to modify the collection object "structurally", a concurrent modification exception will be thrown. It is possible for other threads though to invoke "set" method since it doesn't modify the collection "structurally". However, if prior to calling "set", the collection has been modified structurally, "IllegalArgumentException" will be thrown.

1. **[Why doesn't Collection extend Cloneable and Serializable?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "Collection-extend-Cloneable)**

From Sun FAQ Page: Many Collection implementations (including all of the ones provided by the JDK) will have a public clone method, but it would be mistake to require it of all Collections. For example, what does it mean to clone a Collection that's backed by a terabyte SQL database? Should the method call cause the company to requisition a new disk farm? Similar arguments hold for serializable. If the client doesn't know the actual type of a Collection, it's much more flexible and less error prone to have the client decide what type of Collection is desired, create an empty Collection of this type, and use the addAll method to copy the elements of the original collection into the new one. Note on Some Important Terms

* + Synchronized means only one thread can modify a hash table at one point of time. Basically, it means that any thread before performing an update on a hashtable will have to acquire a lock on the object while others will wait for lock to be released.
  + Fail-fast is relevant from the context of iterators. If an iterator has been created on a collection object and some other thread tries to modify the collection object "structurally”, a concurrent modification exception will be thrown. It is possible for other threads though to invoke "set" method since it doesn’t modify the collection "structurally”. However, if prior to calling "set", the collection has been modified structurally, "IllegalArgumentException" will be thrown.

1. **[How can we make Hashmap synchronized?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "Hashmap-synchronized)**

HashMap can be synchronized by *Map m = Collections.synchronizedMap(hashMap);*

1. **[Where will you use Hashtable and where will you use HashMap?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "Hashtable-vs-Hashmap-use)**

There are multiple aspects to this decision: 1. The basic difference between a Hashtable and an HashMap is that, Hashtable is synchronized while HashMap is not. Thus whenever there is a possibility of multiple threads accessing the same instance, one should use Hashtable. While if not multiple threads are going to access the same instance then use HashMap. Non synchronized data structure will give better performance than the synchronized one. 2. If there is a possibility in future that - there can be a scenario when you may require to retain the order of objects in the Collection with key-value pair then HashMap can be a good choice. As one of HashMap's subclasses is LinkedHashMap, so in the event that you'd want predictable iteration order (which is insertion order by default), you can easily swap out the HashMap for a LinkedHashMap. This wouldn't be as easy if you were using Hashtable. Also if you have multiple thread accessing you HashMap then Collections.synchronizedMap() method can be leveraged. Overall HashMap gives you more flexibility in terms of possible future changes.

1. **[Difference between Vector and ArrayList? What is the Vector class?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "Vector-vs-ArrayList)**

Vector & ArrayList both classes are implemented using dynamically resizable arrays, providing fast random access and fast traversal. ArrayList and Vector class both implement the List interface. Both the classes are member of Java collection framework, therefore from an API perspective, these two classes are very similar. However, there are still some major differences between the two. Below are some key differences

* + Vector is a legacy class which has been retrofitted to implement the List interface since Java 2 platform v1.2
  + Vector is synchronized whereas ArrayList is not. Even though Vector class is synchronized, still when you want programs to run in multithreading environment using ArrayList with Collections.synchronizedList() is recommended over Vector.
  + ArrayList has no default size while vector has a default size of 10.
  + The Enumerations returned by Vector's elements method are not fail-fast. Whereas ArraayList does not have any method returning Enumerations.

1. **[What is the Difference between Enumeration and Iterator interface?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "Difference-between-Enumeration-Iterator)**

Enumeration and Iterator are the interface available in java.util package. The functionality of Enumeration interface is duplicated by the Iterator interface. New implementations should consider using Iterator in preference to Enumeration. Iterators differ from enumerations in following ways:

* + Enumeration contains 2 methods namely hasMoreElements() & nextElement() whereas Iterator contains three methods namely hasNext(), next(),remove().
  + Iterator adds an optional remove operation, and has shorter method names. Using remove() we can delete the objects but Enumeration interface does not support this feature.
  + Enumeration interface is used by legacy classes. Vector.elements() & Hashtable.elements() method returns Enumeration. Iterator is returned by all Java Collections Framework classes. java.util.Collection.iterator() method returns an instance of Iterator.

1. **[Why Java Vector class is considered obsolete or unofficially deprecated? or Why should I always use ArrayList over Vector?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "Why-vector-class-deprecated)**

You should use ArrayList over Vector because you should default to non-synchronized access. Vector synchronizes each individual method. That's almost never what you want to do. Generally you want to synchronize a whole sequence of operations. Synchronizing individual operations is both less safe (if you iterate over a Vector, for instance, you still need to take out a lock to avoid anyone else changing the collection at the same time) but also slower (why take out a lock repeatedly when once will be enough)?  
  
Of course, it also has the overhead of locking even when you don't need to. It's a very flawed approach to have synchronized access as default. You can always decorate a collection using Collections.synchronizedList - the fact that Vector combines both the "resized array" collection implementation with the "synchronize every operation" bit is another example of poor design; the decoration approach gives cleaner separation of concerns.  
  
Vector also has a few legacy methods around enumeration and element retrieval which are different than the List interface, and developers (especially those who learned Java before 1.2) can tend to use them if they are in the code. Although Enumerations are faster, they don't check if the collection was modified during iteration, which can cause issues, and given that Vector might be chosen for its syncronization - with the attendant access from multiple threads, this makes it a particularly pernicious problem. Usage of these methods also couples a lot of code to Vector, such that it won't be easy to replace it with a different List implementation.  
  
Despite all above reasons Sun may never officially deprecate Vector class. (Read details[Deprecate Hashtable and Vector](http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6201870))

1. **[What is an enumeration?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "What-enumeration)**

An enumeration is an interface containing methods for accessing the underlying data structure from which the enumeration is obtained. It is a construct which collection classes return when you request a collection of all the objects stored in the collection. It allows sequential access to all the elements stored in the collection.

1. **[What is the difference between Enumeration and Iterator?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "Enumeration-vs-Iterator)**

The functionality of Enumeration interface is duplicated by the Iterator interface. Iterator has a remove() method while Enumeration doesn't. Enumeration acts as Read-only interface, because it has the methods only to traverse and fetch the objects, where as using Iterator we can manipulate the objects also like adding and removing the objects. So Enumeration is used when ever we want to make Collection objects as Read-only.

1. **[Where will you use Vector and where will you use ArrayList?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "13)**

The basic difference between a Vector and an ArrayList is that, vector is synchronized while ArrayList is not. Thus whenever there is a possibility of multiple threads accessing the same instance, one should use Vector. While if not multiple threads are going to access the same instance then use ArrayList. Non synchronized data structure will give better performance than the synchronized one.

1. **[What is the importance of hashCode() and equals() methods? How they are used in Java?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "14)**

The java.lang.Object has two methods defined in it. They are - public boolean equals(Object obj) public int hashCode(). These two methods are used heavily when objects are stored in collections.  
  
There is a contract between these two methods which should be kept in mind while overriding any of these methods.  
  
The Java API documentation describes it in detail. The hashCode() method returns a hash code value for the object. This method is supported for the benefit of hashtables such as those provided by java.util.Hashtable or java.util.HashMap. The general contract of hashCode is:   
  
Whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same integer, provided no information used in equals comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application. If two objects are equal according to the equals(Object) method, then calling the hashCode method on each of the two objects must produce the same integer result. It is not required that if two objects are unequal according to the equals(java.lang.Object) method, then calling the hashCode method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hashtables. As much as is reasonably practical, the hashCode method defined by class Object does return distinct integers for distinct objects. The equals(Object obj) method indicates whether some other object is "equal to" this one. The equals method implements an equivalence relation on non-null object references:   
  
It is reflexive: for any non-null reference value x, x.equals(x) should return true.   
  
It is symmetric: for any non-null reference values x and y, x.equals(y) should return true if and only if y.equals(x) returns true.   
  
It is transitive: for any non-null reference values x, y, and z, if x.equals(y) returns true and y.equals(z) returns true, then x.equals(z) should return true.   
  
It is consistent: for any non-null reference values x and y, multiple invocations of x.equals(y) consistently return true or consistently return false, provided no information used in equals comparisons on the objects is modified. For any non-null reference value x, x.equals(null) should return false. The equals method for class Object implements the most discriminating possible equivalence relation on objects; that is, for any non-null reference values x and y, this method returns true if and only if x and y refer to the same object (x == y has the value true).  
  
Note that it is generally necessary to override the hashCode method whenever this method is overridden, so as to maintain the general contract for the hashCode method, which states that equal objects must have equal hash codes.  
  
**A practical Example of hashcode() & equals():**This can be applied to classes that need to be stored in Set collections. Sets use equals() to enforce non-duplicates, and HashSet uses hashCode() as a first-cut test for equality. Technically hashCode() isn't necessary then since equals() will always be used in the end, but providing a meaningful hashCode() will improve performance for very large sets or objects that take a long time to compare using equals().

1. **[What is the difference between Sorting performance of Arrays.sort() vs Collections.sort() ? Which one is faster? Which one to use and when?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "15)**

Many developers are concerned about the performance difference between java.util.Array.sort() java.util.Collections.sort() methods. Both methods have same algorithm the only difference is type of input to them. Collections.sort() has a input as List so it does a translation of List to array and vice versa which is an additional step while sorting. So this should be used when you are trying to sort a list. Arrays.sort is for arrays so the sorting is done directly on the array. So clearly it should be used when you have a array available with you and you want to sort it.

1. **[What is java.util.concurrent BlockingQueue? How it can be used?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "16)**

Java has implementation of BlockingQueue available since Java 1.5. Blocking Queue interface extends collection interface, which provides you power of collections inside a queue. Blocking Queue is a type of Queue that additionally supports operations that wait for the queue to become non-empty when retrieving an element, and wait for space to become available in the queue when storing an element. A typical usage example would be based on a producer-consumer scenario. Note that a BlockingQueue can safely be used with multiple producers and multiple consumers. An ArrayBlockingQueue is a implementation of blocking queue with an array used to store the queued objects. The head of the queue is that element that has been on the queue the longest time. The tail of the queue is that element that has been on the queue the shortest time. New elements are inserted at the tail of the queue, and the queue retrieval operations obtain elements at the head of the queue. ArrayBlockingQueue requires you to specify the capacity of queue at the object construction time itself. Once created, the capacity cannot be increased. This is a classic "bounded buffer" (fixed size buffer), in which a fixed-sized array holds elements inserted by producers and extracted by consumers. Attempts to put an element to a full queue will result in the put operation blocking; attempts to retrieve an element from an empty queue will be blocked.

1. **[Set & List interface extend Collection, so Why doesn't Map interface extend Collection?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "17)**

Though the Map interface is part of collections framework, it does not extend collection interface. This is by design, and the answer to this questions is best described in Sun's FAQ Page: This was by design. We feel that mappings are not collections and collections are not mappings. Thus, it makes little sense for Map to extend the Collection interface (or vice versa). If a Map is a Collection, what are the elements? The only reasonable answer is "Key-value pairs", but this provides a very limited (and not particularly useful) Map abstraction. You can't ask what value a given key maps to, nor can you delete the entry for a given key without knowing what value it maps to. Collection could be made to extend Map, but this raises the question: what are the keys? There's no really satisfactory answer, and forcing one leads to an unnatural interface. Maps can be viewed as Collections (of keys, values, or pairs), and this fact is reflected in the three "Collection view operations" on Maps (keySet, entrySet, and values). While it is, in principle, possible to view a List as a Map mapping indices to elements, this has the nasty property that deleting an element from the List changes the Key associated with every element before the deleted element. That's why we don't have a map view operation on Lists.

1. **[Which implementation of the List interface provides for the fastest insertion of a new element into the middle of the list?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "18)**

a. Vector b. ArrayList c. LinkedList ArrayList and Vector both use an array to store the elements of the list. When an element is inserted into the middle of the list the elements that follow the insertion point must be shifted to make room for the new element. The LinkedList is implemented using a doubly linked list; an insertion requires only the updating of the links at the point of insertion. Therefore, the LinkedList allows for fast insertions and deletions.

1. **[What is the difference between ArrayList and LinkedList? (ArrayList vs LinkedList.)](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "19)**

java.util.ArrayList and java.util.LinkedList are two Collections classes used for storing lists of object references **Here are some key differences:**

* + ArrayList uses primitive object array for storing objects whereas LinkedList is made up of a chain of nodes. Each node stores an element and the pointer to the next node. A singly linked list only has pointers to next. A doubly linked list has a pointer to the next and the previous element. This makes walking the list backward easier.
  + ArrayList implements the RandomAccess interface, and LinkedList does not. The commonly used ArrayList implementation uses primitive Object array for internal storage. Therefore an ArrayList is much faster than a LinkedList for random access, that is, when accessing arbitrary list elements using the get method. Note that the get method is implemented for LinkedLists, but it requires a sequential scan from the front or back of the list. This scan is very slow. For a LinkedList, there's no fast way to access the Nth element of the list.
  + Adding and deleting at the start and middle of the ArrayList is slow, because all the later elements have to be copied forward or backward. (Using System.arrayCopy()) Whereas Linked lists are faster for inserts and deletes anywhere in the list, since all you do is update a few next and previous pointers of a node.
  + Each element of a linked list (especially a doubly linked list) uses a bit more memory than its equivalent in array list, due to the need for next and previous pointers.
  + ArrayList may also have a performance issue when the internal array fills up. The arrayList has to create a new array and copy all the elements there. The ArrayList has a growth algorithm of (n\*3)/2+1, meaning that each time the buffer is too small it will create a new one of size (n\*3)/2+1 where n is the number of elements of the current buffer. Hence if we can guess the number of elements that we are going to have, then it makes sense to create a arraylist with that capacity during object creation (using construtor new ArrayList(capacity)). Whereas LinkedLists should not have such capacity issues.

1. **[Where will you use ArrayList and Where will you use LinkedList? Or Which one to use when (ArrayList / LinkedList).](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "20)**

Below is a snippet from SUN's site. The Java SDK contains 2 implementations of the List interface - ArrayList and LinkedList. If you frequently add elements to the beginning of the List or iterate over the List to delete elements from its interior, you should consider using LinkedList. These operations require constant-time in a LinkedList and linear-time in an ArrayList. But you pay a big price in performance. Positional access requires linear-time in a LinkedList and constant-time in an ArrayList.

1. **[What is performance of various Java collection implementations/algorithms? What is Big 'O' notation for each of them ?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "BigONotationJavaCollections)**

Each java collection implementation class have different performance for different methods, which makes them suitable for different programming needs.

## HashTable

## An instance of Hashtable has two parameters that affect its performance: initial capacity and load factor. The capacity is the number of buckets in the hash table, and the initial capacity is simply the capacity at the time the hash table is created. Note that the hash table is open: in the case of a "hash collision", a single bucket stores multiple entries, which must be searched sequentially. The load factor is a measure of how full the hash table is allowed to get before its capacity is automatically increased. The initial capacity and load factor parameters are merely hints to the implementation. The exact details as to when and whether the rehash method is invoked are implementation-dependent.

## Hash Map

## This implementation provides constant-time [ Big O Notation is O(1) ] performance for the basic operations (get and put), assuming the hash function disperses the elements properly among the buckets. Iteration over collection views requires time proportional to the "capacity" of the HashMap instance (the number of buckets) plus its size (the number of key-value mappings). Thus, it's very important not to set the initial capacity too high (or the load factor too low) if iteration performance is important.

## TreeMap

## The TreeMap implementation provides guaranteed log(n) [ Big O Notation is O(log N) ] time cost for the containsKey, get, put and remove operations.

## Linked Hash Map

## A linked hash map has two parameters that affect its performance: initial capacity and load factor. They are defined precisely as for HashMap. Note, however, that the penalty for choosing an excessively high value for initial capacity is less severe for this class than for HashMap, as iteration times for this class are unaffected by capacity.

## Hash Set Map

## The HashSet class offers constant-time [ Big O Notation is O(1) ] performance for the basic operations (add, remove, contains and size), assuming the hash function disperses the elements properly among the buckets. Iterating over this set requires time proportional to the sum of the HashSet instance's size (the number of elements) plus the "capacity" of the backing HashMap instance (the number of buckets). Thus, it's very important not to set the initial capacity too high (or the load factor too low) if iteration performance is important.

## Tree Set

## The TreeSet implementation provides guaranteed log(n) time cost for the basic operations (add, remove and contains).

## Linked Hash Set

## A linked hash set has two parameters that affect its performance: initial capacity and load factor. They are defined precisely as for HashSet. Note, however, that the penalty for choosing an excessively high value for initial capacity is less severe for this class than for HashSet, as iteration times for this class are unaffected by capacity.

## LinkedList

## Performance of get and remove methods is linear time [ Big O Notation is O(n) ] - Performance of add and Iterator.remove methods is constant-time [ Big O Notation is O(1) ]

## ArrayList

## - The size, isEmpty, get, set, iterator, and listIterator operations run in constant time. [ Big O Notation is O(1) ]  - The add operation runs in amortized constant time [ Big O Notation is O(1) ] , but in worst case (since the array must be resized and copied) adding n elements requires linear time [ Big O Notation is O(n) ]  - Performance of remove method is linear time [ Big O Notation is O(n) ]  - All of the other operations run in linear time [ Big O Notation is O(n) ]. The constant factor is low compared to that for the LinkedList implementation.

|  |
| --- |
| Collections Interview Questions |
| **Q1) What is difference between ArrayList and vector?**  Ans: )  1) Synchronization - ArrayList is not thread-safe whereas Vector is thread-safe. In Vector class each method like add(), get(int i) is surrounded with a synchronized block and thus making Vector class thread-safe.  2) Data growth - Internally, both the ArrayList and Vector hold onto their contents using an Array. When an element is inserted into an ArrayList or a Vector, the object will need to expand its internal array if it runs out of room. A Vector defaults to doubling the size of its array, while the ArrayList increases its array size by 50 percent. |
| **Q2) How can Arraylist be synchronized without using Vector?**  Ans) Arraylist can be synchronized using:  Collection.synchronizedList(List list)  Other collections can be synchronized:  Collection.synchronizedMap(Map map)  Collection.synchronizedCollection(Collection c) |
| **Q3) If an Employee class is present and its objects are added in an arrayList. Now I want the list to be sorted on the basis of the employeeID of Employee class. What are the steps?**  Ans) 1) Implement Comparable interface for the Employee class and override the compareTo(Object obj) method in which compare the employeeID  2) Now call Collections.sort() method and pass list as an argument.  Now consider that Employee class is a jar file.  1) Since Comparable interface cannot be implemented, create Comparator and override the compare(Object obj, Object obj1) method .  2) Call Collections.sort() on the list and pass comparator as an argument. |
| **Q4)What is difference between HashMap and HashTable?**  Ans) Both collections implements Map. Both collections store value as key-value pairs. The key differences between the two are  1. Hashmap is not synchronized in nature but hshtable is.  2. Another difference is that iterator in the HashMap is fail-safe while the enumerator for the Hashtable isn't. **Fail-safe** - â€œif the Hashtable is structurally modified at any time after the iterator is created, in any way except through the iterator's own remove method, the iterator will throw a ConcurrentModificationExceptionâ€?  3. HashMap permits null values and only one null key, while Hashtable doesn't allow key or value as null. |
| **Q5) What are the classes implementing List interface?**  Ans)There are three classes that implement List interface: 1) **ArrayList** : It is a resizable array implementation. The size of the ArrayList can be increased dynamically also operations like add,remove and get can be formed once the object is created. It also ensures that the data is retrieved in the manner it was stored. The ArrayList is not thread-safe.  2) **Vector**: It is thread-safe implementation of ArrayList. The methods are wrapped around a synchronized block.  3) **LinkedList**: the LinkedList also implements Queue interface and provide FIFO(First In First Out) operation for add operation. It is faster if than ArrayList if it performs insertion and deletion of elements from the middle of a list. |
| **Q6) Which all classes implement Set interface?**  Ans) A Set is a collection that contains no duplicate elements. More formally, sets contain no pair of elements e1 and e2 such that e1.equals(e2), and at most one null element. **HashSet,SortedSet and TreeSet** are the commnly used class which implements Set interface.  **SortedSet** - It is an interface which extends Set. A the name suggest , the interface allows the data to be iterated in the ascending order or sorted on the basis of Comparator or Comparable interface. All elements inserted into the interface must implement Comparable or Comparator interface.  **TreeSet** - It is the implementation of SortedSet interface.This implementation provides guaranteed log(n) time cost for the basic operations (add, remove and contains). The class is not synchronized.  **HashSet:** This class implements the Set interface, backed by a hash table (actually a HashMap instance). It makes no guarantees as to the iteration order of the set; in particular, it does not guarantee that the order will remain constant over time. This class permits the null element. This class offers constant time performance for the basic operations (add, remove, contains and size), assuming the hash function disperses the elements properly among the buckets |
| **Q7) What is difference between List and a Set?**  Ans) 1) List can contain duplicate values but Set doesnt allow. Set allows only to unique elements.  2) List allows retrieval of data to be in same order in the way it is inserted but Set doesnt ensures the sequence in which data can be retrieved.(Except HashSet) |
| **Q8) What is difference between Arrays and ArrayList ?**  Ans) Arrays are created of fix size whereas ArrayList is of not fix size. It means that once array is declared as :  int [] intArray= new int[6];  intArray[7]   // will give ArraysOutOfBoundException.  Also the size of array cannot be incremented or decremented. But with arrayList the size is variable.Once the array is created elements cannot be added or deleted from it. But with ArrayList the elements can be added and deleted at runtime.  List list = new ArrayList(); list.add(1); list.add(3); list.remove(0) // will remove the element from the 1st location.  ArrayList is one dimensional but array can be multidimensional.  int[][][] intArray= new int[3][2][1];   // 3 dimensional array  To create an array the size should be known or initalized to some value. If not initialized carefully there could me memory wastage. But arrayList is all about dynamic creation and there is no wastage of memory. |
| **Q9) When to use ArrayList or LinkedList ?**  Ans)  Adding new elements is pretty fast for either type of list. For the ArrayList, doing  random lookup using "get" is fast, but for LinkedList, it's slow. It's slow because there's no efficient way to index into the middle of a linked list. When removing elements, using ArrayList is slow. This is because all remaining elements in the underlying array of Object instances must be shifted down for each remove operation. But here LinkedList is fast, because deletion can be done simply by changing a couple of links. So an ArrayList works best for cases where you're doing random access on the list, and a LinkedList works better if you're doing a lot of editing in the middle of the list.  Source : [Read More - from java.sun](http://java.sun.com/developer/TechTips/1999/tt0809.html) |
| **Q10) Consider a scenario. If an ArrayList has to be iterate to read data only, what are the possible ways and which is the fastest?**  Ans) It can be done in two ways, using for loop or using iterator of ArrayList. The first option is faster than using iterator. Because value stored in arraylist is indexed access. So while accessing the value is accessed directly as per the index. |
| **Q11) Now another question with respect to above question is if accessing through iterator is slow then why do we need it and when to use it.**  Ans) For loop does not allow the updation in the array(add or remove operation) inside the loop whereas Iterator does. Also Iterator can be used where there is no clue what type of collections will be used because all collections have iterator. |
| **Q12) Which design pattern Iterator follows?**  Ans) It follows Iterator design pattern. Iterator Pattern is a type of behavioral pattern. The Iterator pattern is one, which allows you to navigate through a collection of data using a common interface without knowing about the underlying implementation. Iterator should be implemented as an interface. This allows the user to implement it anyway its easier for him/her to return data. The benefits of Iterator are about their strength to provide a common interface for iterating through collections without bothering about underlying implementation.  Example of Iteration design pattern - Enumeration The class java.util.Enumeration is an example of the Iterator pattern. It represents and abstract means of iterating over a collection of elements in some sequential order without the client having to know the representation of the collection being iterated over. It can be used to provide a uniform interface for traversing collections of all kinds. |
|  |
| **Q13) Why is it preferred to declare: List<String> list = new ArrayList<String>(); instead of ArrayList<String> = new ArrayList<String>();**  Ans) It is preferred because:   1. If later on code needs to be changed from ArrayList to Vector then only at the declaration place we can do that. 2. The most important one – If a function is declared such that it takes list. E.g void showDetails(List list); When the parameter is declared as List to the function it can be called by passing any subclass of List like ArrayList,Vector,LinkedList making the function more flexible |
| **Q14) What is difference between iterator access and index access?**  Ans) Index based access allow access of the element directly on the basis of index. The cursor of the datastructure can directly goto the 'n' location and get the element. It doesnot traverse through n-1 elements.  In Iterator based access, the cursor has to traverse through each element to get the desired element.So to reach the 'n'th element it need to traverse through n-1 elements.  Insertion,updation or deletion will be faster for iterator based access if the operations are performed on elements present in between the datastructure.  Insertion,updation or deletion will be faster for index based access if the operations are performed on elements present at last of the datastructure.  Traversal or search in index based datastructure is faster.  ArrayList is index access and LinkedList is iterator access. |
| **Q15) How to sort list in reverse order?**  Ans) To sort the elements of the List in the reverse natural order of the strings, get a reverseComparator from the Collections class with reverseOrder(). Then, pass the reverse Comparator to the sort() method.  List list = new ArrayList();  Comparator comp = Collections.reverseOrder();  Collections.sort(list, comp) |
| **Q16) Can a null element added to a Treeset or HashSet?**  Ans) A null element can be added only if the set contains one element because when a second element is added then as per set defination a check is made to check duplicate value and comparison with null element will throw NullPointerException. HashSet is based on hashMap and can contain null element. |
| **Q17) How to sort list of strings - case insensitive?**  Ans) using Collections.sort(list, String.CASE\_INSENSITIVE\_ORDER); |
| **Q18) How to make a List (ArrayList,Vector,LinkedList) read only?**  **Ans) A list implemenation can be made read only using Collections.unmodifiableList(list). This method returns a new list. If a user tries to perform add operation on the new list; UnSupportedOperationException is thrown.** |
| **Q19) What is ConcurrentHashMap?**  **Ans) A concurrentHashMap is thread-safe implementation of Map interface. In this class put and remove method are synchronized but not get method. This class is different from Hashtable in terms of locking; it means that hashtable use object level lock but this class uses bucket level lock thus having better performance.** |
| **Q20) Which is faster to iterate LinkedHashSet or LinkedList?**  **Ans) LinkedList.** |
| **Q21) Which data structure HashSet implements**  **Ans) HashSet implements hashmap internally to store the data. The data passed to hashset is stored as key in hashmap with null as value.** |
| **Q22) Arrange in the order of speed - HashMap,HashTable, Collections.synchronizedMap,concurrentHashmap**  **Ans) HashMap is fastest, ConcurrentHashMap,Collections.synchronizedMap,HashTable.** |
| **Q23) What is identityHashMap?**  **Ans) The IdentityHashMap uses == for equality checking instead of equals(). This can be used for both performance reasons, if you know that two different elements will never be equals and for preventing spoofing, where an object tries to imitate another.** |
| **Q24) What is WeakHashMap?**  **Ans) A hashtable-based Map implementation with weak keys. An entry in a WeakHashMap will automatically be removed when its key is no longer in ordinary use. More precisely, the presence of a mapping for a given key will not prevent the key from being discarded by the garbage collector, that is, made finalizable, finalized, and then reclaimed. When a key has been discarded its entry is effectively removed from the map, so this class behaves somewhat differently than other Map implementations.** |

:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

**What is HashMap and Map?**

Map is Interface and Hashmap is class that implements this interface.

**What is the significance of ListIterator?**

**Or**

**What is the difference b/w Iterator and ListIterator?**

**Iterator :** Enables you to cycle through a collection in the forward direction only, for obtaining or removing elements

**ListIterator :**It extends Iterator, allow bidirectional traversal of list and the modification of elements

**Difference between HashMap and HashTable?** Can we make hashmap synchronized?

1. The HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls. (***HashMap****allows null values as key and value whereas Hashtable doesn’t allow nulls*).  
2. HashMap does not guarantee that the order of the map will remain constant over time.  
3. HashMap is non synchronized whereas Hashtable is synchronized.  
4. Iterator in the HashMap is fail-safe while the enumerator for the Hashtable isn’t.

Note on Some Important Terms  
1)Synchronized means only one thread can modify a hash table at one point of time. Basically, it means that any thread before performing an update on a hashtable will have to acquire a lock on the object while others will wait for lock to be released.

2)Fail-safe is relevant from the context of iterators. If an iterator has been created on a collection object and some other thread tries to modify the collection object “structurally”, a concurrent modification exception will be thrown. It is possible for other threads though to invoke “set” method since it doesn’t modify the collection “structurally”. However, if prior to calling “set”, the collection has been modified structurally, “IllegalArgumentException” will be thrown.

HashMap can be synchronized by

Map m = Collections.synchronizeMap(hashMap);

**What is the difference between set and list?**

A Set stores elements in an unordered way and does not contain duplicate elements, whereas a list stores elements in an ordered way but may contain duplicate elements.

**Difference between Vector and ArrayList? What is the Vector class?**

Vector is synchronized whereas ArrayList is not. The Vector class provides the capability to implement a growable array of objects. ArrayList and Vector class both implement the List interface. Both classes are implemented using dynamically resizable arrays, providing fast random access and fast traversal. In vector the data is retrieved using the elementAt() method while in ArrayList, it is done using the get() method. ArrayList has no default size while vector has a default size of 10. when you want programs to run in multithreading environment then use concept of vector because it is synchronized. But ArrayList is not synchronized so, avoid use of it in a multithreading environment.

**What is an Iterator interface? Is Iterator a Class or Interface? What is its use?**

The Iterator is an interface, used to traverse through the elements of a Collection. It is not advisable to modify the collection itself while traversing an Iterator.

**What is the Collections API?**

The Collections API is a set of classes and interfaces that support operations on collections of objects.  
Example of classes: HashSet, HashMap, ArrayList, LinkedList, TreeSet and TreeMap.  
Example of interfaces: Collection, Set, List and Map.

**What is the List interface?**

The List interface provides support for ordered collections of objects.

**How can we access elements of a collection?**

We can access the elements of a collection using the following ways:  
1.Every collection object has get(index) method to get the element of the object. This method will return Object.  
2.Collection provide Enumeration or Iterator object so that we can get the objects of a collection one by one.

**What is the Set interface?**

The Set interface provides methods for accessing the elements of a finite mathematical set. Sets do not allow duplicate elements.

**What’s the difference between a queue and a stack?**

Stack is a data structure that is based on last-in-first-out rule (LIFO), while queues are based on First-in-first-out (FIFO) rule.

**What is the Map interface?**

The Map interface is used associate keys with values.

**What is the Properties class?**

The properties class is a subclass of Hashtable that can be read from or written to a stream. It also provides the capability to specify a set of default values to be used.

**Which implementation of the List interface provides for the fastest insertion of a new element into the middle of the list?**

a. Vector  
b. ArrayList  
c. LinkedList  
d. None of the above

ArrayList and Vector both use an array to store the elements of the list. When an element is inserted into the middle of the list the elements that follow the insertion point must be shifted to make room for the new element. The LinkedList is implemented using a doubly linked list; an insertion requires only the updating of the links at the point of insertion. Therefore, the LinkedList allows for fast insertions and deletions.

**How can we use hashset in collection interface?**

This class implements the set interface, backed by a hash table (actually a HashMap instance). It makes no guarantees as to the iteration order of the set; in particular, it does not guarantee that the order will remain constant over time. This class permits the Null element.

This class offers constant time performance for the basic operations (add, remove, contains and size), assuming the hash function disperses the elements properly among the buckets.

**What are differences between Enumeration, ArrayList, Hashtable and Collections and Collection?**

Enumeration: It is series of elements. It can be use to enumerate through the elements of a vector, keys or values of a hashtable. You can not remove elements from Enumeration.

ArrayList: It is re-sizable array implementation. Belongs to ‘List’ group in collection. It permits all elements, including null. It is not thread -safe.

Hashtable: It maps key to value. You can use non-null value for key or value. It is part of group Map in collection.

Collections: It implements Polymorphic algorithms which operate on collections.

Collection: It is the root interface in the collection hierarchy.

**What is difference between array & arraylist?**

An ArrayList is resizable, where as, an array is not. ArrayList is a part of the Collection Framework. We can store any type of objects, and we can deal with only objects. It is growable. Array is collection of similar data items. We can have array of primitives or objects. It is of fixed size. We can have multi dimensional arrays.

**Array:** can store primitive **ArrayList:** Stores object only

**Array:** fix size**ArrayList:** resizable

**Array:** can have multi dimensional

**Array:** lang                                **ArrayList:** Collection framework

**Can you limit the initial capacity of vector in java?**

Yes you can limit the initial capacity. We can construct an empty vector with specified initial capacity

public vector(int initialcapacity)

**What method should the key class of Hashmap override?**

The methods to override are equals() and hashCode().

**What is the difference between Enumeration and Iterator?**

The functionality of Enumeration interface is duplicated by the Iterator interface. Iterator has a remove() method while Enumeration doesn’t. Enumeration acts as Read-only interface, because it has the methods only to traverse and fetch the objects, where as using Iterator we can manipulate the objects also like adding and removing the objects.

So Enumeration is used when ever we want to make Collection objects as Read-only.

……………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………. **1) What is the difference between Serializable and Externalizable interface in Java?**

This is most frequently asked question in java serialization interview. Here is my version Externalizable provides us writeExternal () and readExternal () method which gives us flexibility to control java serialization mechanism instead of relying on java's default serialization. Correct implementation of Externalizable interface can improve performance of application drastically.

**2) How many methods Serializable has? If no method then what is the purpose of Serializable interface?**

Serializable interface exists in **java.io**  package and forms core of java serializationmechanism. It doesn't have any method and also called Marker Interface. When your class implements Serializable interface it becomes Serializable in Java and gives compiler an indication that use Java Serialization mechanism to serialize this object.

**3) What is serialVersionUID? What would happen if you don't define this?**

SerialVersionUID is an ID which is stamped on object when it get serialized usually hashcode of object, you can use tool **serialver**to see **serialVersionUID of a serialized object** . serialVersionUID is used for version control of object. you can specify **serialVersionUID**in your class file also.  Consequence of not specifying serialVersionUID is that when you add or modify any field in class then already serialized class will not be able to recover because serialVersionUID generated for new class and for **old serialized object** will be different. Java serialization process relies on correct **serialVersionUID**for recovering state of serialized object and throwsjava.io.InvalidClassException in case of **serialVersionUID**mismatch.

**4) While serializing you want some of the members not to serialize? How do you achieve it?**

this is sometime also asked as what is the use of ***transient variable***, does ***transient and static variable*** gets **serialized** or not etc. so if you don't want any field to be part of object's state then declare it either static or transient based on your need and it will not be included during java serialization process.

**5) What will happen if one of the members in the class doesn't implement Serializable interface?**

If you try to **serialize an object** of a class which implements Serializable, but the object includes a reference to an non- Serializable class then a**‘NotSerializableException’** will be thrown at runtime and this is why I always put a SerializableAlert (comment section in my code) to instruct developer to remember this fact while adding a new field in a Serializable class.

**6) If a class is Serializable but its super class in not, what will be the state of the instance variables inherited from super class after deserialization?**

**Java serialization** process  only continues in object hierarchy till the class is Serializable i.e. implements Serializable interface in Java And values of the instance variables inherited from super class will be initialized by calling constructorof Non-Serializable Super class during deserialization process . once the constructorchaining will started it wouldn't be possible to stop that , hence even if classes higher in hierarchy implements Serializable interface , thereconstructor will be executed.

**7) Can you Customize Serialization process or can you override default Serialization process in Java?**

The answer is yes you can. We all know that *for serializing an object objectOutputStream.writeObject (saveThisobject) is invoked*and for reading object ObjectInputStream.readObject () is invoked but there is one more thing which Java Virtual Machine provides you is to define these two method in your class. If you define these two methods in your class then JVM will invoke these two methods instead of applying default serialization mechanism. You can customize behavior of object serialization or deserialization here by doing any kind of pre or post processing task.Important point to note is making these methods private to avoid being inherited, overridden or overloaded. Since only Java Virtual Machine can call private method integrity of your class will remain and **Java Serialization** will work as normal.

**8) Suppose super class of a new class implement Serializable interface, how can you avoid new class to being serialized?**

If Super Class of a Class already implements Serializable interface in Java then its already serializable in Java, since you can not unimplemented an interface its not really possible to make it Non Serializable class but yes there is a way to avoid serialization of new class. **To avoid java serialization** you need to implement writeObject () and readObject () method in your Class and need to throw**NotSerializableException** from those method. This is another benefit of**customizing java serialization** process as described in above question and normally it asked as follow-up question as interview progresses.

**9) Which methods are used during Serialization and DeSerialization process in java?**

This is quite a common question basically interviewer is trying to know that whether you are familiar with usage of readObject (), writeObject (), readExternal () and writeExternal () or not. Java Serialization is done by java.io.ObjectOutputStream class. That class is a filter stream which is wrapped around a lower-level byte stream to handle the serialization mechanism. To store any object via serialization mechanism we call objectOutputStream.writeObject (saveThisobject) and to deserialize that object we call ObjectInputStream.readObject () method. Call to writeObject () method trigger serialization process in java. one important thing to note about readObject() method is that it is used to read bytes from the persistence and to create object from those bytes and its return an Object which needs to be casted on correct type.

**10) Suppose you have a class which you serialized it and stored in persistence and later modified that class to add a new field. What will happen if you deserialize the object already serialized?**

It depends on whether class has its own **serialVersionUID** or not. As we know from above question that if we don't provide serialVersionUID in our code java compiler will generate it and normally it’s equal to hash code of object. by adding any new field there is chance that new serialVersionUID generated for that class version is not the same of already serialized object and in this case Java Serialization API will throw**java.io.InvalidClassException** and this is the reason **its recommended to haveyour own serialVersionUID** in code and make sure to keep it same always for a single class.

**11) What are the compatible changes and incompatible changes in Java Serialization Mechanism?**

The real challenge lies with change in class structure by adding any field, method or removing any field or method is that with already serialized object***. As per Java Serialization specification adding any field or method comes under compatible change***and changing class hierarchy or unimplementing Serializable interfaces some under non compatible changes. For complete list of compatible and non compatible changes I would advise reading java serialization specification.

**12) Can we transfer a Serialized object vie network?**

*Yes you can transfer a Serialized object via network* because java serialized object remains in form of bytes which can be transmitter via network.

**13) Which kind of variables is not serialized during Java Serialization?**

This question asked sometime differently but the purpose is same whether Java developer knows specifics about ***static and transient variable*** or not. Since *static variables belong to the class* and not to an object they are not the part of the state of object so they are not saved during Java Serialization process. As Java Serialization only persist state of object and not object itself. Transient variables are also not included in java serialization process and are not the part of the object’s serialized state. After this question sometime interviewer ask a follow-up if you don't store values of these variables then what would be value of these variable once you deserialize and recreate those object? This is for you guys to think about :)

**JAVA COLLECTIONS INTERVIEW QUESTIONS AND ANSWERS**  
  
1) Explain Java Collections Framework?   
     Java Collections Framework provides a well designed set of interfaces and classes that support operations on a collections of objects.   
  
2) Explain Iterator Interface.  
     An Iterator is similar to the Enumeration interface.With the Iterator interface methods, you can traverse a collection from start to end and safely remove elements from the underlying Collection. The iterator() method generally used in query operations.  
Basic methods:  
iterator.remove();  
iterator.hasNext();  
iterator.next();   
  
3) Explain Enumeration Interface.  
     The Enumeration interface allows you to iterate through all the elements of a collection. Iterating through an Enumeration is similar to iterating through an Iterator. However, there is no removal support with Enumeration.  
Basic methods:  
boolean hasMoreElements();  
Object nextElement();   
  
4) What is the difference between Enumeration and Iterator interface?  
     The Enumeration interface allows you to iterate through all the elements of a collection. Iterating through an Enumeration is similar to iterating through an Iterator. However, there is no removal support with Enumeration.   
  
5) Explain Set Interface.  
     In mathematical concept, a set is just a group of unique items, in the sense that the group contains no duplicates. The Set interface extends the Collection interface. Set does not allow duplicates in the collection. In Set implementations null is valid entry, but allowed only once.   
  
6) What are the two types of Set implementations available in the Collections Framework?  
     HashSet and TreeSet are the two Set implementations available in the Collections Framework.   
  
7) What is the difference between HashSet and TreeSet?  
     HashSet Class implements java.util.Set interface to eliminate the duplicate entries and uses hashing for storage. Hashing is nothing but mapping between a key value and a data item, this provides efficient searching.  
  
The TreeSet Class implements java.util.Set interface provides an ordered set, eliminates duplicate entries and uses tree for storage.   
  
8) What is a List?  
     List is a ordered and non duplicated collection of objects. The List interface extends the Collection interface.   
  
9) What are the two types of List implementations available in the Collections Framework?  
     ArrayList and LinkedList are the two List implementations available in the Collections Framework.   
  
10) What is the difference between ArrayList and LinkedList?   
     The ArrayList Class implements java.util.List interface and uses array for storage. An array storage's are generally faster but we cannot insert and delete entries in middle of the list.To achieve this kind of addition and deletion requires a new array constructed. You can access any element at randomly.   
  
The LinkedList Class implements java.util.List interface and uses linked list for storage.A linked list allow elements to be added, removed from the collection at any location in the container by ordering the elements.With this implementation you can only access the elements in sequentially.   
  
11) What collection will you use to implement a queue?  
     LinkedList   
  
12) What collection will you use to implement a queue?  
     LinkedList   
  
13) Explain Map Interface.  
     A map is a special kind of set with no duplicates.The key values are used to lookup, or index the stored data. The Map interface is not an extension of Collection interface, it has it's own hierarchy. Map does not allow duplicates in the collection. In Map implementations null is valid entry, but allowed only once.   
  
14) What are the two types of Map implementations available in the Collections Framework?  
     HashMap and TreeMap are two types of Map implementations available in the Collections Framework.   
  
15) What is the difference between HashMap and TreeMap?  
     The HashMap Class implements java.util.Map interface and uses hashing for storage. Indirectly Map uses Set functionality so, it does not permit duplicates. The TreeMap Class implements java.util.Map interface and uses tree for storage. It provides the ordered map.   
  
16) Explain the functionality of Vector Class?  
     Once array size is set you cannot change size of the array. To deal with this kind of situations in Java uses Vector, it grows and shrink it's size automatically. Vector allows to store only objects not primitives. To store primitives, convert primitives in to objects using wrapper classes before adding them into Vector.The Vector reallocates and resizes itself automatically.   
  
17) What does the following statement convey?  
Vector vt = new Vector(3, 10);   
     vt is an instance of Vector class with an initial capacity of 3 and grows in increment of 10 in each relocation   
  
18) How do you store a primitive data type within a Vector or other collections class?   
     You need to wrap the primitive data type into one of the wrapper classes found in the java.lang package, like Integer, Float, or Double, as in:   
Integer in = new Integer(5);   
  
19) What is the difference between Vector and ArrayList?   
     Vector and ArrayList are very similar. Both of them represent a growable array. The main difference is that Vector is synchronized while ArrayList is not.   
  
20) What is the between Hashtable and HashMap?   
     Both provide key-value access to data. The key differences are :  
a. Hashtable is synchronised but HasMap is not synchronised.   
b. HashMap permits null values but Hashtable doent allow null values.  
c. iterator in the HashMap is fail-safe while the enumerator for the Hashtable is not fail safe.   
  
21) How do I make an array larger?  
     You cannot directly make an array larger. You must make a new (larger) array and copy the original elements into it, usually with System.arraycopy(). If you find yourself frequently doing this, the Vector class does this automatically for you, as long as your arrays are not of primitive data types.   
  
22) Which is faster, synchronizing a HashMap or using a Hashtable for thread-safe access?   
     Because a synchronized HashMap requires an extra method call, a Hashtable is faster for synchronized access.   
  
23) In which package would you find the interfaces amd claases defined in the Java Collection Framework?  
     java.util   
  
24) What method in the System class allows you to copy eleemnts from one array to another?  
     System. arraycopy()   
  
25) When using the System.arraycopy() method,What exception is thrown if the destination array is smaller than the souce array?  
     ArrayIndexOutofBoundsException   
  
26) What is the use of Locale class?  
     The Locale class is used to tailor program output to the conventions of a particular geographic, political, or cultural region   
  
27) What is the use of GregorianCalendar class?  
     The GregorianCalendar provides support for traditional Western calendars   
  
28) What is the use of SimpleTimeZone class?  
     The SimpleTimeZone class provides support for a Gregorian calendar   
  
29) What is the use of ResourceBundle class?   
     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.

|  |
| --- |
| Exceptions Interview Questions |
| **Q1) What is an Exception?**  Ans) The exception is said to be thrown whenever an exceptional event occurs in java which signals that something is not correct with the code written and may give unexpected result. An exceptional event is a occurrence of condition which alters the normal program flow. Exceptional handler is the code that does something about the exception. |
| **Q2) Exceptions are defined in which java package?**  Ans)All the exceptions are subclasses of java.lang.Exception |
| **Q3) How are the exceptions handled in java?**  Ans)When an exception occurs the execution of the program is transferred to an appropriate exception handler.The try-catch-finally block is used to handle the exception. The code in which the exception may occur is enclosed in a try block, also called as a guarded region. The catch clause matches a specific exception to a block of code which handles that exception. And the clean up code which needs to be executed no matter the exception occurs or not is put inside the finally block |
| **Q4) Explain the exception hierarchy in java.**  Ans) The hierarchy is as follows:  Throwable is a parent class off all Exception classes. They are two types of Exceptions: Checked exceptions and UncheckedExceptions. Both type of exceptions extends Exception class. |
| **Q5) What is Runtime Exception or unchecked exception?**  Ans) Runtime exceptions represent problems that are the result of a programming problem. Such problems include arithmetic exceptions, such as dividing by zero; pointer exceptions, such as trying to access an object through a null reference; and indexing exceptions, such as attempting to access an array element through an index that is too large or too small. Runtime exceptions need not be explicitly caught in try catch block as it can occur anywhere in a program, and in a typical one they can be very numerous. Having to add runtime exceptions in every method declaration would reduce a program's clarity. Thus, the compiler does not require that you catch or specify runtime exceptions (although you can). The solution to rectify is to correct the programming logic where the exception has occurred or provide a check. |
| **Q6) What is checked exception?**  Ans) Checked exception are the exceptions which forces the programmer to catch them explicitly in try-catch block. It is a subClass of Exception. Example: IOException. |
| **Q7) What is difference between Error and Exception?**  Ans) An error is an irrecoverable condition occurring at runtime. Such as OutOfMemory error. These JVM errors and you can not repair them at runtime.Though error can be caught in catch block but the execution of application will come to a halt and is not recoverable.  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.) |
| **Q8) What is difference between ClassNotFoundException and NoClassDefFoundError?**  Ans) A ClassNotFoundException is thrown when the reported class is not found by the ClassLoader in the CLASSPATH. It could also mean that the class in question is trying to be loaded from another class which was loaded in a parent classloader and hence the class from the child classloader is not visible.  Consider if NoClassDefFoundError occurs which is something like  java.lang.NoClassDefFoundError  src/com/TestClass  does not mean that the TestClass class is not in the CLASSPATH. It means that the class TestClass was found by the ClassLoader however when trying to load the class, it ran into an error reading the class definition. This typically happens when the class in question has static blocks or members which use a Class that's not found by the ClassLoader. So to find the culprit, view the source of the class in question (TestClass in this case) and look for code using static blocks or static members. |
| **Q9) What is throw keyword?**  Ans) Throw keyword is used to throw the exception manually. It is mainly used when the program fails to satisfy the given condition and it wants to warn the application.The exception thrown should be subclass of Throwable.    public void parent(){   try{    child();   }catch(MyCustomException e){ }   }     public void child{   String iAmMandatory=null;    if(iAmMandatory == null){     throw (new MyCustomException("Throwing exception using throw keyword");    }   } |
| **Q10) What is use of throws keyword?**  Ans) If the function is not capable of handling the exception then it can ask the calling method to handle it by simply putting the **throws clause**at the function declaration.    public void parent(){   try{    child();   }catch(MyCustomException e){ }   }     public void child throws MyCustomException{    //put some logic so that the exception occurs.   } |
| **Q11) What are the possible combination to write try, catch finally block?**  Ans)  1) try{  //lines of code that may throw an exception  }catch(Exception e){  //lines of code to handle the exception thrown in try block  }finally{  //the clean code which is executed always no matter the exception occurs or not.  }  2 try{  }finally{}  3 try{  }catch(Exception e){  //lines of code to handle the exception thrown in try block  }  The catch blocks must always follow the try block. If there are more than one catch blocks they all must follow each other without any block in between. The finally block must follow the catch block if one is present or if the catch block is absent the finally block must follow the try block. |
| **Q12) How to create custom Exception?**  Ans) To create you own exception extend the Exception class or any of its subclasses.  e.g.  1 class New1Exception extends Exception { } // this will create Checked Exception  2 class NewException extends IOExcpetion { } // this will create Checked exception  3 class NewException extends NullPonterExcpetion { } // this will create UnChecked exception |
| **Q13) When to make a custom checked Exception or custom unchecked Exception?**  Ans) If an application can reasonably be expected to recover from an exception, make it a checked exception. If an application cannot do anything to recover from the exception, make it an unchecked exception. |
| **Q14)What is StackOverflowError?**  Ans) The StackOverFlowError is an Error Object thorwn by the Runtime System when it Encounters that your application/code has ran out of the memory. It may occur in case of recursive methods or a large amount of data is fetched from the server and stored in some object. This error is generated by JVM.  e.g. void swap(){  swap();  } |
| **Q15) Why did the designers decide to force a method to specify all uncaught checked exceptions that can be thrown within its scope?**  Ans) Any Exception that can be thrown by a method is part of the method's public programming interface. Those who call a method must know about the exceptions that a method can throw so that they can decide what to do about them. These exceptions are as much a part of that method's programming interface as its parameters and return value. |
| **Q16) Once the control switches to the catch block does it return back to the try block to execute the balance code?**  **Ans) No. Once the control jumps to the catch block it never returns to the try block but it goes to finally block(if present).** |
| **Q17) Where is the clean up code like release of resources is put in try-catch-finally block and why?**  **Ans) The code is put in a finally block because irrespective of try or catch block execution the control will flow to finally block. Typically finally block contains release of connections, closing of result set etc.** |
| **Q18) Is it valid to have a try block without catch or finally?**  **Ans) NO. This will result in a compilation error. The try block must be followed by a catch or a finally block. It is legal to omit the either catch or the finally block but not both.**  **e.g. The following code is illegal.**  **try{ int i =0;  } int a = 2; System.out.println(“a = “+a);** |
| **Q19) Is it valid to place some code in between try the catch/finally block that follows it?**  **Ans) No. There should not be any line of code present between the try and the catch/finally block. e.g. The following code is wrong.**  **try{} String str = “ABC”; System.out.println(“str = “+str); catch(Exception e){}** |
| **Q22) Can you catch more than one exceptions in a single catch block?**  **Ans)Yes. If the exception class specified in the catch clause has subclasses, any exception object that is a subclass of the specified Exception class will be caught by that single catch block.**  **E.g..**  **try {**  **// Some code here that can throw an IOException**  **}**  **catch (IOException e) {**  **e.printStackTrace();**  **}**  **The catch block above will catch IOException and all its subclasses e.g. FileNotFoundException etc.** |
| **Q23)Why is not considered as a good practice to write a single catchall handler to catch all the exceptions?**  **Ans) You can write a single catch block to handle all the exceptions thrown during the program execution as follows :**  **try {**  **// code that can throw exception of any possible type**  **}catch (Exception e) {**  **e.printStackTrace();**  **}**  **If you use the Superclass Exception in the catch block then you will not get the valuable information about each of the exception thrown during the execution, though you can find out the class of the exception occurred. Also it will reduce the readability of the code as the programmer will not understand what is the exact reason for putting the try-catch block.** |
| **Q24) What is exception matching?**  **Ans) Exception matching is the process by which the the jvm finds out the matching catch block for the exception thrown from the list of catch blocks. When an exception is thrown, Java will try to find by looking at the available catch clauses in the top down manner. If it doesn't find one, it will search for a handler for a supertype of the exception. If it does not find a catch clause that matches a supertype for the exception, then the exception is propagated down the call stack. This process is called exception matching.** |
| **Q25) What happens if the handlers for the most specific exceptions is placed above the more general exceptions handler?**  **Ans) Compilation fails. The catch block for handling the most specific exceptions must always be placed above the catch block written to handle the more general exceptions.**  **e.g. The code below will not compile.**  **1 try {**  **// code that can throw IOException or its subtypes**  **} catch (IOException e) {**  **// handles IOExceptions and its subtypes**  **} catch (FileNotFoundException ex) {**  **// handle FileNotFoundException only**  **}**  **The code below will compile successfully :-**  **try {  // code that can throw IOException or its subtypes  } catch (FileNotFoundException ex) {  // handles IOExceptions and its subtypes  } catch (IOException e){  // handle FileNotFoundException only  }** |
| **Q26) Does the order of the catch blocks matter if the Exceptions caught by them are not subtype or supertype of each other?**  **Ans) No. If the exceptions are siblings in the Exception class’s hierarchy i.e. If one Exception class is not a subtype or supertype of the other, then the order in which their handlers(catch clauses) are placed does not matter.** |
| **Q27) What happens if a method does not throw an checked Exception directly but calls a method that does? What does 'Ducking' the exception mean?**  **Ans) If a method does not throw an checked Exception directly but calls a method that throws an exception then the calling method must handle the throw exception or declare the exception in its throws clause. If the calling method does not handle and declares the exception, the exceptions is passed to the next method in the method stack. This is called as ducking the exception down the method stack.**  **e.g. The code below will not compile as the getCar() method has not declared the CarNotFoundException which is thrown by the getColor () method.**  **void getCar() {**  **getColor();**  **}**  **void getColor () {**  **throw new CarNotFoundException();**  **}**  **Fix for the above code is**  **void getCar() throws CarNotFoundException {**  **getColor();**  **}**  **void getColor () {**  **throw new CarNotFoundException();**  **}** |
| **Q28) Is an empty catch block legal?**  **Ans) Yes you can leave the catch block without writing any actual code to handle the exception caught.**  **e.g. The code below is legal but not appropriate, as in this case you will nt get any information about the exception thrown.**  **try{**  **//code that may throw the FileNotFoundException**  **}catch(FileNotFound eFnf){**  **//no code to handle the FileNotFound exception**  **}** |
| **Q29)Can a catch block throw the exception caught by itself?**  **Ans) Yes. This is called rethrowing of the exception by catch block.**  **e.g. the catch block below catches the FileNotFound exception and rethrows it again.**  **void checkEx() throws FileNotFoundException {**  **try{**  **//code that may throw the FileNotFoundException**  **}catch(FileNotFound eFnf){**  **throw FileNotFound();**  **}**  **}**  **1) What is an Exception?**  An *exception* is an abnormal condition that arises in a code sequence at run time. In other words, an exception is a run-time error.  **2) What is a Java Exception?**  A Java exception is an object that describes an exceptional condition i.e., an error condition that has occurred in a piece of code. When this type of condition arises, an object representing that exception is created and *thrown* in the method that caused the error by the Java Runtime. That method may choose to handle the exception itself, or pass it on. Either way, at some point, the exception is *caught* and processed.  **3) What are the different ways to generate and Exception?**  There are two different ways to generate an Exception.   1. Exceptions can be generated by the Java run-time system.   Exceptions thrown by Java relate to fundamental errors that violate the rules of the Java language or the constraints of the Java execution environment.   1. Exceptions can be manually generated by your code.   Manually generated exceptions are typically used to report some error condition to the caller of a method.  **4) Where does Exception stand in the Java tree hierarchy?**   * java.lang.**Object** * java.lang.**Throwable** * java.lang.**Exception** * java.lang.**Error**   **5) Is it compulsory to use the finally block?**  It is always a good practice to use the finally block. The reason for using the finally block is, any unreleased resources can be released and the memory can be freed. For example while closing a connection object an exception has occurred. In finally block we can close that object. Coming to the question, you can omit the finally block when there is a catch block associated with that try block. A try block should have at least a catch or a finally block.  **6) How are try, catch and finally block organized?**  A try block should associate with at least a catch or a finally block. The sequence of try, catch and finally matters a lot. If you modify the order of these then the code won’t compile. Adding to this there can be multiple catch blocks associated with a try block. The final concept is there should be a single try, multiple catch blocks and a single finally block in a try-catch-finally block.  **7) What is a throw in an Exception block?**  **“throw”** is used to manually throw an exception (object) of type **Throwable** class or a subclass of **Throwable**. Simple types, such as **int** or **char**, as well as non-**Throwable** classes, such as **String** and **Object**, cannot be used as exceptions. The flow of execution stops immediately after the **throw** statement; any subsequent statements are not executed.  **throw***ThrowableInstance*;  *ThrowableInstance* must be an object of type **Throwable** or a subclass of **Throwable**.  **throw** new NullPointerException("thrownException");  **8) What is the use of throws keyword?**  If a method is capable of causing an exception that it does not handle, it must specify this behavior so that callers of the method can guard themselves against that exception. You do this by including a **throws** clause in the method’s declaration. A **throws** clause lists the types of exceptions that a method might throw.  *type method-name(parameter-list)***throws***exception-list*  {  // body of method  }  **Warning:** main(http://www.javabeat.net/javabeat/templates/faqs/faqs\_middle.html):  failed to open stream: HTTP request failed! HTTP/1.1 404 Not Found in  **/home/content/k/k/s/kkskrishna/html/faqs/exception/exception-faqs-1.html** on line **195**  **Warning:** main(http://www.javabeat.net/javabeat/templates/faqs/faqs\_middle.html):  failed to open stream: HTTP request failed! HTTP/1.1 404 Not Found in  **/home/content/k/k/s/kkskrishna/html/faqs/exception/exception-faqs-1.html** on line **195**  **Warning:** main(): Failed opening 'http://www.javabeat.net/javabeat/templates/faqs/faqs\_middle.html' for inclusion  (include\_path='.:/usr/local/lib/php') in **/home/content/k/k/s/kkskrishna/html/faqs/exception/exception-faqs-1.html**  on line **195**  Here, *exception-list* is a comma-separated list of the exceptions that a method can throw.  static void throwOne() **throws** IllegalAccessException {  System.out.println("Inside throwOne.");  **9) What are Checked Exceptions and Unchecked Exceptions?**  The types of exceptions that need not be included in a methods **throws** list are called **Unchecked Exceptions**.   * ArithmeticException * ArrayIndexOutOfBoundsException * ClassCastException * IndexOutOfBoundsException * IllegalStateException * NullPointerException * SecurityException   The types of exceptions that must be included in a methods **throws** list if that method can generate one of these exceptions and does not handle it itself are called **Checked Exceptions**.   * ClassNotFoundException * CloneNotSupportedException * IllegalAccessException * InstantiationException * InterruptedException * NoSuchFieldException * NoSuchMethodException   **10) What are Chained Exceptions?**  The chained exception feature allows you to associate another exception with an exception. This second exception describes the cause of the first exception. Lets take a simple example. You are trying to read a number from the disk and using it to divide a number. Think the method throws an **ArithmeticException** because of an attempt to divide by zero (number we got). However, the problem was that an I/O error occurred, which caused the divisor to be set improperly (set to zero). Although the method must certainly throw an **ArithmeticException**, since that is the error that occurred, you might also want to let the calling code know that the underlying cause was an I/O error. This is the place where chained exceptions come in to picture.  Throwable getCause( )    Throwable initCause(Throwable causeExc)   * 1. Which package contains exception handling related classes?  java.lang   2) What are the two types of Exceptions?   Checked Exceptions and Unchecked Exceptions.   3) What is the base class of all exceptions?   java.lang.Throwable   4) What is the difference between Exception and Error in java?   Exception and Error are the subclasses of the Throwable class. Exception class is used for exceptional conditions that user program should catch. Error defines exceptions that are not excepted to be caught by the user program. Example is Stack Overflow.   5) What is the difference between throw and throws?   Throw is used to explicitly raise a exception within the program, the statement would be throw new Exception(); throws clause is used to indicate the exceptions that are not handled by the method. It must specify this behavior so the callers of the method can guard against the exceptions.  Throws is specified in the method signature. If multiple exceptions are not handled, then they are separated by a comma. the statement would be as follows: public void doSomething() throws IOException,MyException{}   6) Differentiate between Checked Exceptions and Unchecked Exceptions?   Checked Exceptions are those exceptions which should be explicitly handled by the calling method. Unhandled checked exceptions results in compilation error.   Unchecked Exceptions are those which occur at runtime and need not be explicitly handled. RuntimeException and it's subclasses, Error and it's subclasses fall under unchecked exceptions.   7) What are User defined Exceptions?   Apart from the exceptions already defined in Java package libraries, user can define his own exception classes by extending Exception class.   8) What is the importance of finally block in exception handling?   Finally block will be executed whether or not an exception is thrown. If an exception is thrown, the finally block will execute even if no catch statement match the exception. Any time a method is about to return to the caller from inside try/catch block, via an uncaught exception or an explicit return statement, the finally block will be executed. Finally is used to free up resources like database connections, IO handles, etc.   9) Can a catch block exist without a try block?   No. A catch block should always go with a try block.   10) Can a finally block exist with a try block but without a catch?   Yes. The following are the combinations try/catch or try/catch/finally or try/finally.   11) What will happen to the Exception object after exception handling?   Exception object will be garbage collected.   12) The subclass exception should precede the base class exception when used within the catch clause. True/False?   True.   13) Exceptions can be caught or rethrown to a calling method. True/False?   True.   14) The statements following the throw keyword in a program are not executed. True/False?   True.   15) How does finally block differ from finalize() method?   Finally block will be executed whether or not an exception is thrown. So it is used to free resoources. finalize() is a protected method in the Object class which is called by the JVM just before an object is garbage collected.   16) What are the constraints imposed by overriding on exception handling?   An overriding method in a subclass may only throw exceptions declared in the parent class or children of the exceptions declared in the parent class.  Java Exceptions Questions **Explain the user defined Exceptions?**  User defined Exceptions are custom Exception classes defined by the user for specific purpose. A user defined exception can be created by simply sub-classing an Exception class or a subclass of an Exception class. This allows custom exceptions to be generated (using throw clause) and caught in the same way as normal exceptions. **Example:**  class CustomException extends Exception {  }  **What classes of exceptions may be caught by a catch clause?**  A catch clause can catch any exception that may be assigned to the Throwable type. This includes the Error and Exception types. Errors are generally irrecoverable conditions  **What is the difference between exception and error?**  Error's are irrecoverable exceptions. Usually a program terminates when an error is encountered.  **What is the difference between throw and throws keywords?**  The throw keyword denotes a statement that causes an exception to be initiated. It takes the Exception object to be thrown as an argument. The exception will be caught by an enclosing try-catch block or propagated further up the calling hierarchy. The throws keyword is a modifier of a method that denotes that an exception may be thrown by the method. An exception can be rethrown.  **What class of exceptions are generated by the Java run-time system?**  The Java runtime system generates Runtime Exceptions and Errors.  **What is the base class for Error and Exception?**  Throwable  **What are Checked and Unchecked Exceptions?**  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 exception may be thrown. Checked exceptions must be caught at compile time. Example: IOException. 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. Example: ArrayIndexOutOfBoundsException. Errors are often irrecoverable conditions.  **Does the code in finally block get executed if there is an exception and a return statement in a catch block?**  **Or**  **What is the purpose of the finally clause of a try-catch-finally statement?**  The finally clause is used to provide the capability to execute code no matter whether or not an exception is thrown or caught. If an exception occurs and there is a return statement in catch block, the finally block is still executed. The finally block will not be executed when the System.exit(0) statement is executed earlier or on system shut down earlier or the memory is used up earlier before the thread goes to finally block.  try{ //some statements } catch{ //statements when exception is caught } finally{ //statements executed whether exception occurs or not }  **Does the order of placing catch statements matter in the catch block?**  Yes, it does. The FileNoFoundException is inherited from the IOException. So FileNoFoundException is caught before IOException. Exception’s subclasses have to be caught first before the General Exception  **Q: What do you mean by exception and error handling?**  A: When a program encounters and un-expected situation from where it cannot continue the normal flow, it can break from the normal execution flow by throwing and exception or an error.  **Q: What is the difference between an Error and an Exception?**  A: An Exception is an unexpected scenario from which the program can recover but an error means that the program has encountered an unrecoverable problem.  **Q: Can you give an example of an unrecoverable problem when error is thrown?**  A: Yes, when a java program runs out of memory it is a problem from which the program cannot recover and OutOfMemoryError will be thrown.  **Q: Does it mean that when error is thrown, we cannot catch it and continue execution?**  A: We can catch errors, but it is not advised to do that because when error happens it means that some unrecoverable problem has happened. Even if we catch the error, we cannot guarantee the stability of the application and something else might fail.  **Q: What are the different kinds of Exceptions?**  A: Checked Exceptions and Unchecked Exceptions.  **Q: What are runtime Exceptions?**  A: Un Checked Exceptions are also known as Runtime Exceptions.  **Q: What do you mean by Checked Exceptions?**  A: Checked exceptions are those exceptions which have to be explicitly handled in the code.  **Q: Can we create our own checked exceptions?**  A: Yes, we just have to extend the Exception class or any of its sub classes (except RuntimeException)  **Q: How can we create our own runtime exception?**  A: We have to extend the RuntimeException class or any of its sub classes.  **Q: How can we create our own Errors?**  A: We have to extend the Error class.  **Q: What are the different approaches of Exception handling?**  A: We can use Try Catch block or we can declare the exception in the method definition's throws clause  **Q: Explain the difference between the two approaches of exception handling**  A: We use the try catch clause where we want to process the exception and perform some action like displaying an error message. When an exception is caught using the catch clause, it will not be propagated to the caller.  We declare the exception in the throws clause of the calling method when the exception has to be propagated to the caller and the exception is supposed to be handled by the caller.  **Q: What is finally?**  A: Finally is also a block like catch block to be used with the try block. Control will come to finally block irrespective of whether exception was thrown or not.  **Q: What is the difference between catch and finally?**  A: When a catch block is used, we have to mention what exception has to be caught in the catch clause; where as in finally we need not mention anything. Code in the catch block is executed only when the exception is thrown whereas the code in finally is executed irrespective of whether exception was thrown or not.  **Q: Can we have a try block without a catch block?**  A: yes, then we must have a finally block. When try block is used, we must use either a catch block or a finally block or both.  **Q: What happens when we use only try and finally block without a catch block?**  A: When catch block is not used, the exceptions that are thrown will not be caught and the exception will be propagated to the caller, but before the exception is propagated, the code in the finally block will be executed.  **Q: When do you use a catch block and when do you use a finally block?**  A: We use a catch block when we want to handle the exception scenario. We use a finally block alone when we want to do some cleanup but at the same time propagate the exception to the caller. We use a finally block together with a catch block when we want to do something irrespective of what exception is thrown.  **Q: what will happen if we return from the try block, will the finally block get executed?**  A: yes, after the return statement in the try block is executed, the statements in the finally block will be executed before the control goes to the calling method.  **Q: What happens when we have a return statement in the try block as well as in the finally block.**  A: when we have return statement both in try and finally blocks, then first the return statement of try block gets executed, the before the method returns, the statements in the finally block will get executed and since there is return statement in finally also, that will also get executed and the value that will be returned to the caller will be the value returned from the finally block.  **Q: If I write System.exit (0); at the end of the try block, will the finally block still execute?**  A: No.  **Q: Can I have more than one catch block following a try block?**  A: yes,  **Q: what are the rules for having multiple catch blocks?**  A:  <!--[if !supportLists]-->1) <!--[endif]-->Each exception can be caught only once.  <!--[if !supportLists]-->2) <!--[endif]-->The catch block for sub class exceptions should come before e the catch block for parent class exceptions for example, FileNotFoundException extends IOException, so the catch block for FileNotFoundException should come before the catch block for IOException.  **Q: Suppose I have a class A that has a method X which throws FileNotFoundException. I have a class B that extends class A and the method X is overridden in B. In the ovverriden method X in class B can I throw IOException?**  A: No, because FileNotFoundException is a subclass if IOException and in the overridden method X in class B we can throw only FileNotFoundException or any other exception that extends FileNotFoundException.  **Q: Is the above rule logical?**  A: yes, the above rule is logical because java supports runtime polymorphism and at runtime an object of type B can be assigned to a variable of type A because B extends A, and when the method X is called on the variable of type A, the compile only check that the calling method should either catch or declare the exception that method X in class A is throwing, and if the method X in class B throws IOException, and the caller is handling only FileNotFoundException, then the exception will escape unhandled.  This message contains information that may be privileged or confidential and is the property of the Capgemini Group. It is intended only for the person to whom it is addressed. If you are not the intended recipient, you are not authorized to read, print, retain, copy, disseminate, distribute, or use this message or any part thereof. If you receive this message in error, please notify the sender immediately and delete all copies of this message. |

**How HashMap works in Java**  
How HashMap works in Java or sometime how get method work in HashMap iscommon interview questions now days. Almost everybody who worked in Java knows what hashMap is, where to use hashMap or difference between hashtable and HashMap then why this interview questionbecomes so special? Because of the breadth and depth this question offers. It has become very popular java interview question in almost any senior or mid-senior level java interviews.  
  
Questions start with simple statement   
  
**"Have you used HashMap before" or "What is HashMap? Why do we use it “**  
Almost everybody answers this with yes and then interviewee keep talking about common facts about hashMap like [hashMap accpt null while hashtable doesn't](http://www.blogger.com/),[HashMap is not synchronized](http://javarevisited.blogspot.com/2010/10/difference-between-hashmap-and.html), hashMap is fast and so on along with basics like its stores key and value pairs etc.  
This shows that person has used hashMap and quite familier with the funtionalities HashMap offers but interview takes a sharp turn from here and next set of follow up questions gets more detailed about fundamentals involved in hashmap. Interview here you and come back with questions like  
  
**"Do you Know how hashMap works in Java”** or   
**"How does get () method of HashMap works in Java"**  
And then you get answers like I don't bother its standard Java API, you better look code on java; I can find it out in Google at any time etc.  
But some interviewee definitely answer this and will say **"HashMap works onprinciple of hashing**, we have put () and get () method for storing and retrieving data from hashMap. When we pass an object to put () method to store it on hashMap, hashMap implementation calls  
hashcode() method hashMap key object and by applying that hashcode on its own hashing funtion it identifies a bucket location for storing value object , important part here is HashMap stores both key+value in bucket which is essential to understand the retrieving logic. if people fails to recognize this and say it only stores Value in the bucket they will fail to explain the retrieving logic of any object stored in HashMap . This answer is very much acceptable and does make sense that interviewee has fair bit of knowledge how hashing works and how HashMap works in Java.  
But this is just start of story and going forward when depth increases a little bit and when you put interviewee on scenarios every java developers faced day by day basis. So next question would be more likely about collision detection and collision resolution in Java HashMap e.g   
  
**"What will happen if two different objects have same hashcode?”**  
Now from here confusion starts some time interviewer will say that since Hashcode is equal objects are equal and HashMap will throw exception or not store it again etc. then you might want to remind them aobut equals and hashCode() contract that two unequal object in Java very much can have equal hashcode. Some will give up at this point and some will move ahead and say "Since hashcode () is same, bucket location would be same and collision occurs in hashMap, Since HashMap use a linked list to store in bucket, value object will be stored in next node of linked list." great this answer make sense to me though there could be some other collision resolution methods available this is simplest and HashMap does follow this.  
But story does not end here and final questions interviewer ask like   
  
**"How will you retreive if two different objects have same hashcode?”**  
 Hmmmmmmmmmmmmm  
Interviewee will say we will call get() method and then HashMap uses keys hashcode to find out bucket location and retreives object but then you need to remind him that there are two objects are stored in same bucket , so they will say about traversal in linked list until we find the value object , then you ask **how do you identify vlaue object because you don't value object to compare** ,So until they know that HashMap stores both Key and Value in linked list node they won't be able to resolve this issue and will try and fail.  
  
But those bunch of people who remember this key information will say that after finding bucket location , we will **call keys.equals() method** to identify correct node in linked list and return associated value object for that key in Java HashMap. Perfect this is the correct answer.  
  
In many cases interviewee fails at this stage because they **get confused between hashcode () and equals ()** and keys and values object in hashMap which is pretty obvious because they are dealing with the hashcode () in all previous questions and equals () come in picture only in case of retrieving value object from HashMap.  
Some good developer point out here that using immutable, final object with proper equals () and hashcode () implementation would act as perfect Java HashMap keys and**improve performance of Java hashMap by reducing collision**. I*mmutablity also allows caching there hashcode of different keys* which makes overall retreival process very fast and suggest that String and various wrapper classes e.g Integer provided by Java Collection API are very good HashMap keys.  
  
Now if you clear all this java hashmap interview question you will be surprised by this very interesting question "**What happens On HashMap in Java if the size of the Hashmap exceeds a given threshold defined by load factor ?"**. Until you know how hashmap works exactly you won't be able to answer this question.   
if the size of the map exceeds a given threshold defined by load-factor e.g. if load factor is .75 it will act to re-size the map once it filled 75%. Java Hashmap does that by creating another new bucket array of size twice of previous size of hashmap, and then start putting every old element into that new bucket array and this process is called rehashing because it also applies hash function to find new bucket location.   
  
If you manage to answer this question on hashmap in java you will be greeted by **"do you see any problem with resizing of hashmap in Java"**, you might not be able to pick the context and then he will try to give you hint about multiple thread accessing the java hashmap and potentially looking for **race condition on HashMap in Java**.   
  
So the answer is Yes there is potential race condition exists while resizing hashmap in Java, if two thread at the same time found that now Java Hashmap needs resizing and they both try to resizing. on the process of **resizing of hashmap in Java** , the element in bucket which is stored in linked list get reversed in order during there migration to new bucket because java hashmap doesn't append the new element at tail instead it append new element at head **to avoid tail traversing**. if race condition happens then you will end up with an infinite loop. though this point you can potentially argue that what the hell makes you think to use HashMap in multi-threaded environment to interviewer :)   
  
I like this question because of its depth and number of concept it touches indirectly, if you look at questions asked during interview this HashMap questions has verified   
*Concept of hashing*  
*Collision resolution in HashMap*  
*Use of equals () and hashCode () method and there importance?*  
*Benefit of immutable object?*  
*race condition on hashmap in Java*  
*Resizing of Java HashMap*  
  
Just to summararize here are the answers which does makes sense for above questions  
  
**How HashMAp works in Java**  
HashMap works on principle of hashing, we have put () and get () method for storing and retrieving object form hashMap.When we pass an both key and value to put() method to store on HashMap, it uses key object hashcode() method to calculate hashcode and they by applying hashing on that hashcode it identifies bucket location for storing value object.  
While retrieving it uses key object equals method to find out correct key value pair and return value object associated with that key. HashMap uses linked list in case of collision and object will be stored in next node of linked list.  
Also hashMap stores both key+value tuple in every node of linked list.  
  
**What will happen if two different HashMap key objects have same hashcode?**  
They will be stored in same bucket but no next node of linked list. And keys equals () method will be used to identify correct key value pair in HashMap.  
  
In terms of usage HashMap is very versatile and I have mostly used hashMap as cache in electronic trading application I have worked . Since finance domain used Java heavily and due to performance reason we need caching a lot HashMap comes as very handy there.

Collections classes are heart of java API though I feel using them judiuously is an art.its my personal experience where I have improved performance byusing ArrayList where legacy codes are unnecesarily used Vectoretc.  
  
JDK 1.5 introduce some good concurrent collections which is highly efficient for high volume , low latency system.  
  
The synchronized collections classes, Hashtable and Vector, and the synchronizedwrapper classes, Collections.synchronizedMap and Collections.synchronizedList, provide a basic conditionally thread-safe implementation of Map and List.  
However, several factors make them unsuitable for use in highly concurrentapplications -- their single collection-wide lock is an impediment to scalability and it often becomes necessary to lock a collection for a considerable time during iteration to prevent ConcurrentModificationExceptions.  
  
The ConcurrentHashMap and CopyOnWriteArrayList implementations provide much higher concurrency while preserving thread safety, with some minor compromises in their promises to callers. ConcurrentHashMap and CopyOnWriteArrayList are not necessarily useful everywhere you might use HashMap or ArrayList, but are designed to optimize specific common situations. Many concurrent applications will benefit from their use.  
  
So what is the difference between hashtable and ConcurrentHashMap , both can be used in multithreaded environment but once the size of hashtable becomes considerable large performance degrade because for iteration it has to be locked for longer duration.  
  
Since ConcurrentHashMap indroduced concept of segmentation , how large it becomes only certain part of it get locked to provide thread safety so many other readers can still access map without waiting for iteration to complete.  
  
In Summary ConcurrentHashMap only locked certain portion of Map while Hashtable lock full map while doing iteration.

///////////////////

question oftenly asked in interview to check whether candidate understand correctusage of collection classes and aware of alternative solutions available.

1. The HashMap class is roughly equivalent to Hashtable, except that it is non synchronized and permits nulls. (HashMap allows null values as key and value whereas Hashtable doesn't allow nulls).

2. HashMap does not guarantee that the order of the map will remain constant over time.

3. HashMap is non synchronized whereas Hashtable is synchronized.

4. Iterator in the HashMap is  fail-fast  while the enumerator for the Hashtable is not and throw ConcurrentModificationException if any other Thread modifies the map structurally  by adding or removing any element except Iterator's own remove()  method. But this is not a guaranteed behavior and will be done by JVM on best effort.

Note on Some Important Terms

1)Synchronized means only one thread can modify a hash table at one point of time. Basically, it means that any thread before performing an update on a hashtable will have to acquire a lock on the object while others will wait for lock to be released.

2)Fail-safe is relevant from the context of iterators. If an iterator has been created on a collection object and some other thread tries to modify the collection object "structurally", a concurrent modification exception wjavascript:void(0)ill be thrown. It is possible for other threads though to invoke "set" method since it doesn't modify the collection "structurally". However, if prior to calling "set", the collection has been modified structurally, "IllegalArgumentException" will be thrown.

3)Structurally modification means deleting or inserting element which could effectively change the structure of map.

HashMap can be synchronized by

Map m = Collections.synchronizeMap(hashMap);

# Introduction to hashing

On this and the following pages, we explore **hashing**, a technique used (among other applications) to implement Java's common [map](http://www.javamex.com/tutorials/collections/using_4.shtml) and [set](http://www.javamex.com/tutorials/collections/using_3.shtml) classes. Hashing is designed to solve the problem of needing to **efficiently** find or store an item in a collection. For example, if we have a list of 10,000 words of English and we want to check if a given word is in the list, it would be inefficient to successively compare the word with all 10,000 items until we find a match. Hashing is a technique to make things more efficient by effectively narrowing down the search at the outset.

## What is hashing?

Hashing means using some function or algorithm to map object data to some representative integer value. This so-called **hash code** (or simply **hash**) can then be used as a way to **narrow down our search** when looking for the item in the map.

## How hashing works

Purely as an example to help us grasp the concept, let's suppose that we want to map a list of string keys to string values (for example, map a list of countries to their capital cities). So let's say we want to store the data in Table 1 in the map.

|  |  |
| --- | --- |
| Table 1: Example data to put in a map | |
| **Key** | **Value** |
| Cuba | Havana |
| England | London |
| France | Paris |
| Spain | Madrid |
| Switzerland | Berne |

And let's suppose that our **hash function** is to simply take the length of the string. For simplicity, we'll have two arrays: one for our keys and one for the values. So to put an item in the hash table, we compute its hash code (in this case, simply count the number of characters), then put the key and value in the arrays at the corresponding index. For example, Cuba has a hash code (length) of 4. So we store Cuba in the 4th position in the keys array, andHavana in the 4th index of the values array etc. And we end up with the following:

|  |  |  |
| --- | --- | --- |
| **Position (hash code = key length)** | **Keys array** | **Values array** |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 | Cuba | Havana |
| 5 | Spain | Madrid |
| 6 | France | Paris |
| 7 | England | London |
| 8 |  |  |
| 9 |  |  |
| 10 |  |  |
| 11 | Switzerland | Berne |

Now, in this specific example things work quite well. Our array needs to be big enough to accommodate the longest string, but in this case that's only 11 slots. And we do waste a bit of space because, for example, there's no 1-letter keys in our data, nor keys between 8 and 10 letters. But in this case, the waste isn't so bad either. And taking the length of a string is nice and fast, so so is the process of finding the value associated with a given key (certainly faster than doing up to five string comparisons)1.

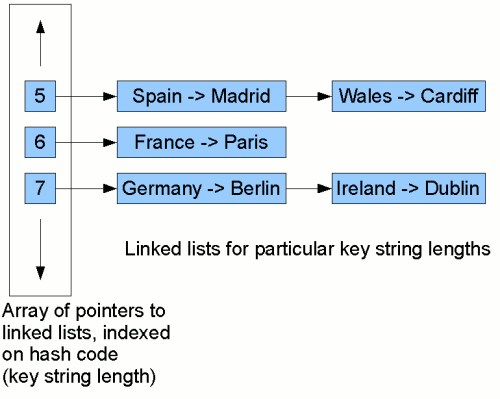
We can also easily see that this method **wouldn't work for storing arbitrary strings**. If one of our string keys was a thousand characters in length but the rest were small, we'd waste the majority of the space in the arrays. More seriously, this model can't deal with **collisions**: that is, what to do when there is more than one key with the same hash code (in this case, one than more string of a given length). For example, if our keys were random words of English, taking the string length would be fairly useless. Granted, the word psuedoantidisestablishmentarianistically would probably get its own place in the array. But on the other hand, we'd be left with thousands of, say, 6-letter words all competing for the same slot in the array.

On the previous page, we introduced the notion of [hashing](http://www.javamex.com/tutorials/collections/hashing_intro.shtml), mapping a piece of data such as a string to some kind of a representative integer value. We can then create a map by using this hash as an index into an array of key/value pairs. Such a structure is generally called a **hash table** or, particularly in Java parlance, **hash map**1. We saw that using the string length to create the hash, and indexing a simple array, could work in some restricted cases, but is no good generally: for example, we have the problem of **collisions** (several keys with the same length) and wasted space if a few keys are vastly larger than the majority.

### Buckets

Now, we can solve the problem of **collisions** by having an array of (references to) **linked lists**2 rather than simply an array of keys/values. Each little list is generally called a **bucket**.

Then, we can solve the problem of having an array that is too large simply by taking the hash code modulo a certain array size3. So for example, if the array were 32 positions in size, going from 0-31, then rather than storing a key/value pair in the list at position 33, we store it at position (33 mod 32) = 1. (In simple terms, we "wrap round" when we reach the end of the array.) So we end up with a structure something like this:



Each node in the linked lists stores a pairing of a key with a value. Now, to look for the mapping for, say, Ireland, we first compute this key's hash code (in this case, the string length, 7). Then we start traversing the linked list at position 7 in the table. We traverse each node in the list, comparing the key stored in that node with Ireland. When we find a match, we return the value from the pair stored in that node (Dublin). In our example here, we find it on the second comparison. So although we have to do some comparisons, if the list at a given position in the table is fairly short, we'll still reduce significantly the amount of work we need to do to find a given key/value mapping.

The structure we have just illustrated is essentially the one used by Java's hash maps and hash sets. However, we generally wouldn't want to use the string length as the hash code. In the next sections, we'll explore how to generate more adequate hash codes.

## Improving our hash function

The method that we use to turn an object into a hash code is called the **hash function**. We'll see on the next page that rather than using the string length, we need to [use a more adequate hash function](http://www.javamex.com/tutorials/collections/hashmaps3.shtml).

Notes:  
1. In the Java community, this double nomenclature has come about for historic reasons. Originally, Java had the class Hashtable with all-synchronized methods. When the Java Collections framework was introduced, Hashtable was effectively replaced with the HashMap class. As with collections classes in general, this class has unsynchronized methods, and via the method Collections.synchronizedMap() can be "wrapped" in a synchronized accessors.  
2. This is the way Java hash maps solve the problem and is the avenue we will explore for the remainder of this tutorial. Another method sometimes used is to store colliding keys/values in positions in the array adjacent to where they would otherwise have gone. This of course requires that the array is big enough to hold as many mappings as will be added.  
3. In other words, we divide the hash code by the array size and use the remainder as the array position.

# ///////////////////////////////////////////////////////////////////////////////////////////////////////////////

# [equals( ) Versus == in Java](http://www.java-samples.com/showtutorial.php?tutorialid=221)

By: Mashoud Emailed: 289 times Printed: 337 times

It is important to understand that the **equals( )**method and the **==**operator perform two different operations. As just explained, the **equals( )**method compares the characters inside a **String**object. The **==**operator compares two object references to see whether they refer to the same instance. The following program shows how two different **String**objects can contain the same characters, but references to these objects will not compare as equal:

**// equals() vs ==   
class EqualsNotEqualTo {   
public static void main(String args[]) {   
String s1 = "Hello";   
String s2 = new String(s1);   
System.out.println(s1 + " equals " + s2 + " -> " +   
s1.equals(s2));   
System.out.println(s1 + " == " + s2 + " -> " + (s1 == s2));   
}   
}**

The variable **s1**refers to the **String**instance created by "**Hello**". The object referred to by **s2**is created with **s1**as an initializer. Thus, the contents of the two **String**objects are identical, but they are distinct objects. This means that **s1**and **s2**do not refer to the same objects and are, therefore, not **==**, as is shown here by the output of the preceding example:

Hello equals Hello -> true   
Hello == Hello -> false

# *==, .equals(), compareTo(), and compare()*

## Equality comparison: One way for primitives, Four ways for objects

|  |  |  |
| --- | --- | --- |
| Comparison | Primitives | Objects |
| a == b, a != b | Equal values | **Compares references, not values.** The use of == with object references is generally limited to the following:   * Comparing to see if a reference is null. * Comparing two enum values. This works because there is only one object for each enum constant. * You want to know if two references are to the *same object* |
| a.equals(b) | N/A | Compares values for equality. Because this method is defined in the Object class, from which all other classes are derived, it's automatically defined for every class. However, it doesn't perform an intelligent comparison for most classes unless the class overrides it. It has been defined in a meaningful way for most Java core classes. If it's not defined for a (user) class, it behaves the same as ==.  It turns out that defining equals() isn't trivial; in fact it's moderately hard to get it right, especially in the case of subclasses. The best treatment of the issues is in Horstmann's *Core Java Vol 1*. [TODO: Add explanation and example] |
| a.compareTo(b) | N/A | **Comparable interface.** Compares values and returns an int which tells if the values compare less than, equal, or greater than. If your class objects have a natural order, implement the *Comparable<T>* interface and define this method. All Java classes that have a natural ordering implement this (String, Double, BigInteger, ...). |
| compare(a, b) | N/A | **Comparator interface.** Compares values of two objects. This is implemented as part of the *Comparator<T>* interface, and the typical use is to define one or more small utility classes that implement this, to pass to methods such assort() or for use by sorting data structures such as TreeMap and TreeSet. You might want to create a Comparator object for the following.   * **Multiple comparisons.** To provide several different ways to sort something. For example, you might want to sort a Person class by name, ID, age, height, ... You would define a Comparator for each of these to pass to the sort() method. * **System class.** To provide comparison methods for classes that you have no control over. For example, you could define a Comparator for Strings that compared them by length. * **Strategy pattern.** To implement a *strategy* pattern, which is a situation where you want to represent an *algorithm* as an object that you can pass as a parameter, save in a data structure, etc.   If your class objects have one natural sorting order, you may not need this. |

## Comparing Object references with the == and != Operators

The two operators that can be used with object references are comparing for equality (**==**) and inequality (**!=**). These operators compare two values to see if they **refer to the same object**. Although this comparison is very fast, it is often not what you want.

Usually you want to know if the objects have the same *value*, and not whether two objects are a *reference* to the same object. For example,

if (name == "Mickey Mouse") // Legal, but ALMOST SURELY WRONG

This is true only if name is a reference to the *same object* that "Mickey Mouse" refers to. This will be false if the String in name was read from input or computed (by putting strings together or taking the substring), even though name really does have exactly those characters in it.

Many classes (eg, String) define the equals() method to compare the *values* of objects.

## Comparing Object values with the equals() Method

Use the equals() method to compare object values. The equals() method returns a boolean value. The previous example can be fixed by writing:

if (name.**equals**("Mickey Mouse")) // Compares values, not references.

Because the equals() method makes a == test first, it can be fairly fast when the objects are identical. It only compares the values if the two references are not identical.

## Other comparisons - Comparable<T> interface

The equals method and == and != operators test for equality/inequality, but do not provide a way to test for relative values. Some classes (eg, String and other classes with a natural ordering) implement the*Comparable<T>* interface, which defines a compareTo method. You will want to implement *Comparable<T>* in your class if you want to use it with Collections.sort() or Arrays.sort() methods.

## Defining a Comparator object

As described in the table above on compare(), you can create Comparators to sort any arbitrary way for any class. For example, the String class defines the CASE\_INSENSITIVE\_ORDER comparator.

## If you override equals, you should also override hashCode()

**Overriding hashCode()**. The hashCode() method of a class is used for *hashing* in library data structures such as HashSet and HashMap. If you override equals(), you should override hashCode() or your class will not work correctly in these (and some other) data structures.

## Shouldn't .equals and .compareTo produce same result?

The general advice is that if a.equals(b) is true, then a.compareTo(b) == 0 should also be true. Curiously,BigDecimal violates this. Look at the Java API documentation for an explanation of the difference. This seems wrong, although their implementation has some plausibility.

## Other comparison methods

String has the specialized equalsIgnoreCase() and compareToIgnoreCase(). String also supplies the constantString.CASE\_INSENSITIVE\_ORDER Comparator.

## The === operator (Doesn't exist - yet?)

Comparing objects is somewhat awkward, so a === operator has been proposed. One proposal is that   
a === b would be the same as ((a == b) || ((a != null) && a.equals(b)))

## Common Errors

**Using == instead of equals() with Objects**

When you want to compare objects, you need to know whether you should use == to see if they are the*same object*, or equals() to see if they may be a different object, but have the *same value*. This kind of error can be very hard to find.

# *Aggregation*

Aggregation is a relationship between two classes that is best described as a "has-a" and "whole/part" relationship. It is a more specialized version of the [association relationship](http://java.about.com/od/a/g/association.htm). The aggregate class contains a reference to another class and is said to have ownership of that class. Each class referenced is considered to be *part-of* the aggregate class

Ownership occurs because there can be no cyclic references in an aggregation relationship. If Class A contains a reference to Class B and Class B contains a reference to Class A then no clear ownership can be determined and the relationship is simply one of association.

For example, imagine a Student class that stores information about individual students at a school. Now let's say there is a Subject class that holds the details about a particular subject (e.g., history, geography). If the Student class is defined to contain a Subject object then it can be said that the Student object *has-a* Subject object. The Subject object also makes up *part-of* the Student object, after all there is no student without a subject to study. The Student object is therefore the owner of the Subject object.

**Examples:**

There is an aggregation relationship between Student class and the Subject class:

public class Subject {

  private String name;

  public void setName(String name)

  {

    this.name = name;

  }

  public String getName()

  {

    return name;

  }

}

public class Student {

  private Subject[] studyAreas = new Subject[10];

  //the rest of the Student class

}

# Association, Aggregation, Composition, Abstraction, Generalization, Realization, Dependency

26/06/2010

These terms signify the relationships between classes. These are the building blocks of object oriented programming and very basic stuff. But still for some, these terms look like Latin and Greek. Just wanted to refresh these terms and explain in simpler terms.

## Association

Association is a relationship between two objects. In other words, association defines the multiplicity between objects. You may be aware of one-to-one, one-to-many, many-to-one, many-to-many all these words define an association between objects. Aggregation is a special form of association. Composition is a special form of aggregation.

http://javapapers.com/wp-content/uploads/2010/06/association.jpg

**Example:**A Student and a Faculty are having an association.

## Aggregation

Aggregation is a special case of association. A directional association between objects. When an object ‘has-a’ another object, then you have got an aggregation between them. Direction between them specified which object contains the other object. Aggregation is also called a “Has-a” relationship.

http://javapapers.com/wp-content/uploads/2010/06/aggregation.jpg

## Composition

Composition is a special case of aggregation. In a more specific manner, a restricted aggregation is called composition. When an object contains the other object, if the contained object cannot exist without the existence of container object, then it is called composition.

http://javapapers.com/wp-content/uploads/2010/06/composition.jpg

**Example:**A class contains students. A student cannot exist without a class. There exists composition between class and students.

### Difference between aggregation and composition

Composition is more restrictive. When there is a composition between two objects, the composed object cannot exist without the other object. This restriction is not there in aggregation. Though one object can contain the other object, there is no condition that the composed object must exist. The existence of the composed object is entirely optional. In both aggregation and composition, direction is must. The direction specifies, which object contains the other object.

***Example:***A Library contains students and books. Relationship between library and student is aggregation. Relationship between library and book is composition. A student can exist without a library and therefore it is aggregation. A book cannot exist without a library and therefore its a composition. For easy understanding I am picking this example. Don’t go deeper into example and justify relationships!

## Abstraction

Abstraction is specifying the framework and hiding the implementation level information. Concreteness will be built on top of the abstraction. It gives you a blueprint to follow to while implementing the details. Abstraction reduces the complexity by hiding low level details.

***Example:***A wire frame model of a car.

## Generalization

Generalization uses a “is-a” relationship from a specialization to the generalization class. Common structure and behaviour are used from the specializtion to the generalized class. At a very broader level you can understand this as inheritance. Why I take the term inheritance is, you can relate this term very well. Generalization is also called a “Is-a” relationship.

http://javapapers.com/wp-content/uploads/2010/06/generalization.jpg

***Example:*** Consider there exists a class named Person. A student is a person. A faculty is a person. Therefore here the relationship between student and person, similarly faculty and person is generalization.

## Realization

Realization is a relationship between the blueprint class and the object containing its respective implementation level details. This object is said to realize the blueprint class. In other words, you can understand this as the relationship between the interface and the implementing class.

http://javapapers.com/wp-content/uploads/2010/06/realization.jpg

***Example:*** A particular model of a car ‘GTB Fiorano’ that implements the blueprint of a car realizes the abstraction.

## Dependency

Change in structure or behaviour of a class affects the other related class, then there is a dependency between those two classes. It need not be the same vice-versa. When one class contains the other class it this happens.

http://javapapers.com/wp-content/uploads/2010/06/dependency.jpg

***Example:***Relationship between shape and circle is dependency.

1. [**What is Java Collections API?**](http://www.fromdev.com/2008/05/java-collections-questions.html#Java-Collections-API)

Java Collections framework API is a unified architecture for representing and manipulating collections. The API contains Interfaces, Implementations & Algorithm to help java programmer in everyday programming. In nutshell, this API does 6 things at high level

* + Reduces programming efforts. - Increases program speed and quality.
  + Allows interoperability among unrelated APIs.
  + Reduces effort to learn and to use new APIs.
  + Reduces effort to design new APIs.
  + Encourages & Fosters software reuse.

To be specific, There are six collection java interfaces. The most basic interface is Collection. Three interfaces extend Collection: Set, List, and SortedSet. The other two collection interfaces, Map and SortedMap, do not extend Collection, as they represent mappings rather than true collections.

1. [**What is an Iterator?**](http://www.fromdev.com/2008/05/java-collections-questions.html#What-is-Iterator)

Some of the collection classes provide traversal of their contents via a java.util.Iterator interface. This interface allows you to walk through a collection of objects, operating on each object in turn. Remember when using Iterators that they contain a snapshot of the collection at the time the Iterator was obtained; generally it is not advisable to modify the collection itself while traversing an Iterator.

1. [**What is the difference between java.util.Iterator and java.util.ListIterator?**](http://www.fromdev.com/2008/05/java-collections-questions.html#Iterator-vs-ListIterator)

Iterator : Enables you to traverse through a collection in the forward direction only, for obtaining or removing elements ListIterator : extends Iterator, and allows bidirectional traversal of list and also allows the modification of elements.

1. [**What is HashMap and Map?**](http://www.fromdev.com/2008/05/java-collections-questions.html#What-is-HashMap)

Map is Interface which is part of Java collections framework. This is to store Key Value pair, and Hashmap is class that implements that using hashing technique.

1. [**Difference between HashMap and HashTable? Compare Hashtable vs HashMap?**](http://www.fromdev.com/2008/05/java-collections-questions.html#HashMap-vs-HashTable)

Both Hashtable & HashMap provide key-value access to data. The Hashtable is one of the original collection classes in Java (also called as legacy classes). HashMap is part of the new Collections Framework, added with Java 2, v1.2. There are several differences between HashMap and Hashtable in Java as listed below

* + The HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls. (HashMap allows null values as key and value whereas Hashtable doesn’t allow nulls).
  + HashMap does not guarantee that the order of the map will remain constant over time. But one of HashMap's subclasses is LinkedHashMap, so in the event that you'd want predictable iteration order (which is insertion order by default), you can easily swap out the HashMap for a LinkedHashMap. This wouldn't be as easy if you were using Hashtable.
  + HashMap is non synchronized whereas Hashtable is synchronized.
  + Iterator in the HashMap is fail-fast while the enumerator for the Hashtable isn't. So this could be a design consideration.

1. [**What does synchronized means in Hashtable context?**](http://www.fromdev.com/2008/05/java-collections-questions.html#synchronized-Hashtable)

Synchronized means only one thread can modify a hash table at one point of time. Any thread before performing an update on a hashtable will have to acquire a lock on the object while others will wait for lock to be released.

1. [**What is fail-fast property?**](http://www.fromdev.com/2008/05/java-collections-questions.html#fail-fast-property)

At high level - Fail-fast is a property of a system or software with respect to its response to failures. A fail-fast system is designed to immediately report any failure or condition that is likely to lead to failure. Fail-fast systems are usually designed to stop normal operation rather than attempt to continue a possibly-flawed process. When a problem occurs, a fail-fast system fails immediately and visibly. Failing fast is a non-intuitive technique: "failing immediately and visibly" sounds like it would make your software more fragile, but it actually makes it more robust. Bugs are easier to find and fix, so fewer go into production. In Java, Fail-fast term can be related to context of iterators. If an iterator has been created on a collection object and some other thread tries to modify the collection object "structurally", a concurrent modification exception will be thrown. It is possible for other threads though to invoke "set" method since it doesn't modify the collection "structurally". However, if prior to calling "set", the collection has been modified structurally, "IllegalArgumentException" will be thrown.

1. [**Why doesn't Collection extend Cloneable and Serializable?**](http://www.fromdev.com/2008/05/java-collections-questions.html#Collection-extend-Cloneable)

From Sun FAQ Page: Many Collection implementations (including all of the ones provided by the JDK) will have a public clone method, but it would be mistake to require it of all Collections. For example, what does it mean to clone a Collection that's backed by a terabyte SQL database? Should the method call cause the company to requisition a new disk farm? Similar arguments hold for serializable. If the client doesn't know the actual type of a Collection, it's much more flexible and less error prone to have the client decide what type of Collection is desired, create an empty Collection of this type, and use the addAll method to copy the elements of the original collection into the new one. Note on Some Important Terms

* + Synchronized means only one thread can modify a hash table at one point of time. Basically, it means that any thread before performing an update on a hashtable will have to acquire a lock on the object while others will wait for lock to be released.
  + Fail-fast is relevant from the context of iterators. If an iterator has been created on a collection object and some other thread tries to modify the collection object "structurally”, a concurrent modification exception will be thrown. It is possible for other threads though to invoke "set" method since it doesn’t modify the collection "structurally”. However, if prior to calling "set", the collection has been modified structurally, "IllegalArgumentException" will be thrown.

1. [**How can we make Hashmap synchronized?**](http://www.fromdev.com/2008/05/java-collections-questions.html#Hashmap-synchronized)

HashMap can be synchronized by *Map m = Collections.synchronizedMap(hashMap);*

1. [**Where will you use Hashtable and where will you use HashMap?**](http://www.fromdev.com/2008/05/java-collections-questions.html#Hashtable-vs-Hashmap-use)

There are multiple aspects to this decision: 1. The basic difference between a Hashtable and an HashMap is that, Hashtable is synchronized while HashMap is not. Thus whenever there is a possibility of multiple threads accessing the same instance, one should use Hashtable. While if not multiple threads are going to access the same instance then use HashMap. Non synchronized data structure will give better performance than the synchronized one. 2. If there is a possibility in future that - there can be a scenario when you may require to retain the order of objects in the Collection with key-value pair then HashMap can be a good choice. As one of HashMap's subclasses is LinkedHashMap, so in the event that you'd want predictable iteration order (which is insertion order by default), you can easily swap out the HashMap for a LinkedHashMap. This wouldn't be as easy if you were using Hashtable. Also if you have multiple thread accessing you HashMap then Collections.synchronizedMap() method can be leveraged. Overall HashMap gives you more flexibility in terms of possible future changes.

1. [**Difference between Vector and ArrayList? What is the Vector class?**](http://www.fromdev.com/2008/05/java-collections-questions.html#Vector-vs-ArrayList)

Vector & ArrayList both classes are implemented using dynamically resizable arrays, providing fast random access and fast traversal. ArrayList and Vector class both implement the List interface. Both the classes are member of Java collection framework, therefore from an API perspective, these two classes are very similar. However, there are still some major differences between the two. Below are some key differences

* + Vector is a legacy class which has been retrofitted to implement the List interface since Java 2 platform v1.2
  + Vector is synchronized whereas ArrayList is not. Even though Vector class is synchronized, still when you want programs to run in multithreading environment using ArrayList with Collections.synchronizedList() is recommended over Vector.
  + ArrayList has no default size while vector has a default size of 10.
  + The Enumerations returned by Vector's elements method are not fail-fast. Whereas ArraayList does not have any method returning Enumerations.

1. [**What is the Difference between Enumeration and Iterator interface?**](http://www.fromdev.com/2008/05/java-collections-questions.html#Difference-between-Enumeration-Iterator)

Enumeration and Iterator are the interface available in java.util package. The functionality of Enumeration interface is duplicated by the Iterator interface. New implementations should consider using Iterator in preference to Enumeration. Iterators differ from enumerations in following ways:

* + Enumeration contains 2 methods namely hasMoreElements() & nextElement() whereas Iterator contains three methods namely hasNext(), next(),remove().
  + Iterator adds an optional remove operation, and has shorter method names. Using remove() we can delete the objects but Enumeration interface does not support this feature.
  + Enumeration interface is used by legacy classes. Vector.elements() & Hashtable.elements() method returns Enumeration. Iterator is returned by all Java Collections Framework classes. java.util.Collection.iterator() method returns an instance of Iterator.

1. [**Why Java Vector class is considered obsolete or unofficially deprecated? or Why should I always use ArrayList over Vector?**](http://www.fromdev.com/2008/05/java-collections-questions.html#Why-vector-class-deprecated)

You should use ArrayList over Vector because you should default to non-synchronized access. Vector synchronizes each individual method. That's almost never what you want to do. Generally you want to synchronize a whole sequence of operations. Synchronizing individual operations is both less safe (if you iterate over a Vector, for instance, you still need to take out a lock to avoid anyone else changing the collection at the same time) but also slower (why take out a lock repeatedly when once will be enough)? Of course, it also has the overhead of locking even when you don't need to. It's a very flawed approach to have synchronized access as default. You can always decorate a collection using Collections.synchronizedList - the fact that Vector combines both the "resized array" collection implementation with the "synchronize every operation" bit is another example of poor design; the decoration approach gives cleaner separation of concerns. Vector also has a few legacy methods around enumeration and element retrieval which are different than the List interface, and developers (especially those who learned Java before 1.2) can tend to use them if they are in the code. Although Enumerations are faster, they don't check if the collection was modified during iteration, which can cause issues, and given that Vector might be chosen for its syncronization - with the attendant access from multiple threads, this makes it a particularly pernicious problem. Usage of these methods also couples a lot of code to Vector, such that it won't be easy to replace it with a different List implementation. Despite all above reasons Sun may never officially deprecate Vector class. (Read details [Deprecate Hashtable and Vector](http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6201870))

1. [**What is an enumeration?**](http://www.fromdev.com/2008/05/java-collections-questions.html#What-enumeration)

An enumeration is an interface containing methods for accessing the underlying data structure from which the enumeration is obtained. It is a construct which collection classes return when you request a collection of all the objects stored in the collection. It allows sequential access to all the elements stored in the collection.

1. [**What is the difference between Enumeration and Iterator?**](http://www.fromdev.com/2008/05/java-collections-questions.html#Enumeration-vs-Iterator)

The functionality of Enumeration interface is duplicated by the Iterator interface. Iterator has a remove() method while Enumeration doesn't. Enumeration acts as Read-only interface, because it has the methods only to traverse and fetch the objects, where as using Iterator we can manipulate the objects also like adding and removing the objects. So Enumeration is used when ever we want to make Collection objects as Read-only.

1. [**Where will you use Vector and where will you use ArrayList?**](http://www.fromdev.com/2008/05/java-collections-questions.html#13)

The basic difference between a Vector and an ArrayList is that, vector is synchronized while ArrayList is not. Thus whenever there is a possibility of multiple threads accessing the same instance, one should use Vector. While if not multiple threads are going to access the same instance then use ArrayList. Non synchronized data structure will give better performance than the synchronized one.

1. [**What is the importance of hashCode() and equals() methods? How they are used in Java?**](http://www.fromdev.com/2008/05/java-collections-questions.html#14)

The java.lang.Object has two methods defined in it. They are - public boolean equals(Object obj) public int hashCode(). These two methods are used heavily when objects are stored in collections. There is a contract between these two methods which should be kept in mind while overriding any of these methods. The Java API documentation describes it in detail. The hashCode() method returns a hash code value for the object. This method is supported for the benefit of hashtables such as those provided by java.util.Hashtable or java.util.HashMap. The general contract of hashCode is: Whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same integer, provided no information used in equals comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application. If two objects are equal according to the equals(Object) method, then calling the hashCode method on each of the two objects must produce the same integer result. It is not required that if two objects are unequal according to the equals(java.lang.Object) method, then calling the hashCode method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hashtables. As much as is reasonably practical, the hashCode method defined by class Object does return distinct integers for distinct objects. The equals(Object obj) method indicates whether some other object is "equal to" this one. The equals method implements an equivalence relation on non-null object references: It is reflexive: for any non-null reference value x, x.equals(x) should return true. It is symmetric: for any non-null reference values x and y, x.equals(y) should return true if and only if y.equals(x) returns true. It is transitive: for any non-null reference values x, y, and z, if x.equals(y) returns true and y.equals(z) returns true, then x.equals(z) should return true. It is consistent: for any non-null reference values x and y, multiple invocations of x.equals(y) consistently return true or consistently return false, provided no information used in equals comparisons on the objects is modified. For any non-null reference value x, x.equals(null) should return false. The equals method for class Object implements the most discriminating possible equivalence relation on objects; that is, for any non-null reference values x and y, this method returns true if and only if x and y refer to the same object (x == y has the value true). Note that it is generally necessary to override the hashCode method whenever this method is overridden, so as to maintain the general contract for the hashCode method, which states that equal objects must have equal hash codes. **A practical Example of hashcode() & equals():**This can be applied to classes that need to be stored in Set collections. Sets use equals() to enforce non-duplicates, and HashSet uses hashCode() as a first-cut test for equality. Technically hashCode() isn't necessary then since equals() will always be used in the end, but providing a meaningful hashCode() will improve performance for very large sets or objects that take a long time to compare using equals().

1. [**What is the difference between Sorting performance of Arrays.sort() vs Collections.sort() ? Which one is faster? Which one to use and when?**](http://www.fromdev.com/2008/05/java-collections-questions.html#15)

Many developers are concerned about the performance difference between java.util.Array.sort() java.util.Collections.sort() methods. Both methods have same algorithm the only difference is type of input to them. Collections.sort() has a input as List so it does a translation of List to array and vice versa which is an additional step while sorting. So this should be used when you are trying to sort a list. Arrays.sort is for arrays so the sorting is done directly on the array. So clearly it should be used when you have a array available with you and you want to sort it.

1. [**What is java.util.concurrent BlockingQueue? How it can be used?**](http://www.fromdev.com/2008/05/java-collections-questions.html#16)

Java has implementation of BlockingQueue available since Java 1.5. Blocking Queue interface extends collection interface, which provides you power of collections inside a queue. Blocking Queue is a type of Queue that additionally supports operations that wait for the queue to become non-empty when retrieving an element, and wait for space to become available in the queue when storing an element. A typical usage example would be based on a producer-consumer scenario. Note that a BlockingQueue can safely be used with multiple producers and multiple consumers. An ArrayBlockingQueue is a implementation of blocking queue with an array used to store the queued objects. The head of the queue is that element that has been on the queue the longest time. The tail of the queue is that element that has been on the queue the shortest time. New elements are inserted at the tail of the queue, and the queue retrieval operations obtain elements at the head of the queue. ArrayBlockingQueue requires you to specify the capacity of queue at the object construction time itself. Once created, the capacity cannot be increased. This is a classic "bounded buffer" (fixed size buffer), in which a fixed-sized array holds elements inserted by producers and extracted by consumers. Attempts to put an element to a full queue will result in the put operation blocking; attempts to retrieve an element from an empty queue will be blocked.

1. [**Set & List interface extend Collection, so Why doesn't Map interface extend Collection?**](http://www.fromdev.com/2008/05/java-collections-questions.html#17)

Though the Map interface is part of collections framework, it does not extend collection interface. This is by design, and the answer to this questions is best described in Sun's FAQ Page: This was by design. We feel that mappings are not collections and collections are not mappings. Thus, it makes little sense for Map to extend the Collection interface (or vice versa). If a Map is a Collection, what are the elements? The only reasonable answer is "Key-value pairs", but this provides a very limited (and not particularly useful) Map abstraction. You can't ask what value a given key maps to, nor can you delete the entry for a given key without knowing what value it maps to. Collection could be made to extend Map, but this raises the question: what are the keys? There's no really satisfactory answer, and forcing one leads to an unnatural interface. Maps can be viewed as Collections (of keys, values, or pairs), and this fact is reflected in the three "Collection view operations" on Maps (keySet, entrySet, and values). While it is, in principle, possible to view a List as a Map mapping indices to elements, this has the nasty property that deleting an element from the List changes the Key associated with every element before the deleted element. That's why we don't have a map view operation on Lists.

1. [**Which implementation of the List interface provides for the fastest insertion of a new element into the middle of the list?**](http://www.fromdev.com/2008/05/java-collections-questions.html#18)

a. Vector b. ArrayList c. LinkedList ArrayList and Vector both use an array to store the elements of the list. When an element is inserted into the middle of the list the elements that follow the insertion point must be shifted to make room for the new element. The LinkedList is implemented using a doubly linked list; an insertion requires only the updating of the links at the point of insertion. Therefore, the LinkedList allows for fast insertions and deletions.

1. [**What is the difference between ArrayList and LinkedList? (ArrayList vs LinkedList.)**](http://www.fromdev.com/2008/05/java-collections-questions.html#19)

java.util.ArrayList and java.util.LinkedList are two Collections classes used for storing lists of object references **Here are some key differences:**

* + ArrayList uses primitive object array for storing objects whereas LinkedList is made up of a chain of nodes. Each node stores an element and the pointer to the next node. A singly linked list only has pointers to next. A doubly linked list has a pointer to the next and the previous element. This makes walking the list backward easier.
  + ArrayList implements the RandomAccess interface, and LinkedList does not. The commonly used ArrayList implementation uses primitive Object array for internal storage. Therefore an ArrayList is much faster than a LinkedList for random access, that is, when accessing arbitrary list elements using the get method. Note that the get method is implemented for LinkedLists, but it requires a sequential scan from the front or back of the list. This scan is very slow. For a LinkedList, there's no fast way to access the Nth element of the list.
  + Adding and deleting at the start and middle of the ArrayList is slow, because all the later elements have to be copied forward or backward. (Using System.arrayCopy()) Whereas Linked lists are faster for inserts and deletes anywhere in the list, since all you do is update a few next and previous pointers of a node.
  + Each element of a linked list (especially a doubly linked list) uses a bit more memory than its equivalent in array list, due to the need for next and previous pointers.
  + ArrayList may also have a performance issue when the internal array fills up. The arrayList has to create a new array and copy all the elements there. The ArrayList has a growth algorithm of (n\*3)/2+1, meaning that each time the buffer is too small it will create a new one of size (n\*3)/2+1 where n is the number of elements of the current buffer. Hence if we can guess the number of elements that we are going to have, then it makes sense to create a arraylist with that capacity during object creation (using construtor new ArrayList(capacity)). Whereas LinkedLists should not have such capacity issues.

1. [**Where will you use ArrayList and Where will you use LinkedList? Or Which one to use when (ArrayList / LinkedList).**](http://www.fromdev.com/2008/05/java-collections-questions.html#20)

Below is a snippet from SUN's site. The Java SDK contains 2 implementations of the List interface - ArrayList and LinkedList. If you frequently add elements to the beginning of the List or iterate over the List to delete elements from its interior, you should consider using LinkedList. These operations require constant-time in a LinkedList and linear-time in an ArrayList. But you pay a big price in performance. Positional access requires linear-time in a LinkedList and constant-time in an ArrayList.

1. [**What is performance of various Java collection implementations/algorithms? What is Big 'O' notation for each of them ?**](http://www.fromdev.com/2008/05/java-collections-questions.html#BigONotationJavaCollections)

Each java collection implementation class have different performance for different methods, which makes them suitable for different programming needs.

## Performance of Map interface implementations

## Hashtable

An instance of Hashtable has two parameters that affect its performance: initial capacity and load factor. The capacity is the number of buckets in the hash table, and the initial capacity is simply the capacity at the time the hash table is created. Note that the hash table is open: in the case of a "hash collision", a single bucket stores multiple entries, which must be searched sequentially. The load factor is a measure of how full the hash table is allowed to get before its capacity is automatically increased. The initial capacity and load factor parameters are merely hints to the implementation. The exact details as to when and whether the rehash method is invoked are implementation-dependent.

## HashMap

This implementation provides constant-time [ Big O Notation is O(1) ] performance for the basic operations (get and put), assuming the hash function disperses the elements properly among the buckets. Iteration over collection views requires time proportional to the "capacity" of the HashMap instance (the number of buckets) plus its size (the number of key-value mappings). Thus, it's very important not to set the initial capacity too high (or the load factor too low) if iteration performance is important.

## TreeMap

The TreeMap implementation provides guaranteed log(n) [ Big O Notation is O(log N) ] time cost for the containsKey, get, put and remove operations.

## LinkedHashMap

A linked hash map has two parameters that affect its performance: initial capacity and load factor. They are defined precisely as for HashMap. Note, however, that the penalty for choosing an excessively high value for initial capacity is less severe for this class than for HashMap, as iteration times for this class are unaffected by capacity.

## Performance of Set interface implementations

## HashSet

The HashSet class offers constant-time [ Big O Notation is O(1) ] performance for the basic operations (add, remove, contains and size), assuming the hash function disperses the elements properly among the buckets. Iterating over this set requires time proportional to the sum of the HashSet instance's size (the number of elements) plus the "capacity" of the backing HashMap instance (the number of buckets). Thus, it's very important not to set the initial capacity too high (or the load factor too low) if iteration performance is important.

## TreeSet

The TreeSet implementation provides guaranteed log(n) time cost for the basic operations (add, remove and contains).

## LinkedHashSet

A linked hash set has two parameters that affect its performance: initial capacity and load factor. They are defined precisely as for HashSet. Note, however, that the penalty for choosing an excessively high value for initial capacity is less severe for this class than for HashSet, as iteration times for this class are unaffected by capacity.

## Performance of List interface implementations

## LinkedList

- Performance of get and remove methods is linear time [ Big O Notation is O(n) ] - Performance of add and Iterator.remove methods is constant-time [ Big O Notation is O(1) ]

## ArrayList

- The size, isEmpty, get, set, iterator, and listIterator operations run in constant time. [ Big O Notation is O(1) ] - The add operation runs in amortized constant time [ Big O Notation is O(1) ] , but in worst case (since the array must be resized and copied) adding n elements requires linear time [ Big O Notation is O(n) ] - Performance of remove method is linear time [ Big O Notation is O(n) ] - All of the other operations run in linear time [ Big O Notation is O(n) ]. The constant factor is low compared to that for the LinkedList implementation.

All the differences between LinkedList and ArrayList has there root on difference between [Array](http://javarevisited.blogspot.com/2012/01/anonymous-array-example-java-create.html) and LinkedList data-structure. If you are familiar with Array and LinkedList data structure you will most likely derive following differences between them:

1) Since Array is an index based data-structure searching or getting element from Array with index is pretty fast. Array provides O(1) performance for get(index) method but remove is costly in ArrayList as you need to rearrange all elements. On the Other hand LinkedList doesn't provide Random or index based access and you need to iterate over linked list to retrieve any element which is of order O(n).

2) Insertions  are easy and fast in LinkedList as compared to ArrayList because there is no risk of resizing array

and copying content to new array if array gets full which makes adding into ArrayList of O(n) in worst case, while adding is O(1) operation in LinkedList in Java. ArrayList also needs to update its index if you insert something anywhere except at the end of array.

3) Removal is like insertions better in LinkedList than ArrayList.

4) LinkedList has more memory overhead than ArrayList because in ArrayList each index only holds actual object (data) but in case of LinkedList each node holds both data and address of next  and previous node.

**HashMap** and Hashtable both implement **java.util.Map** interface but there are some differences that Java developers must understand to write more efficient code. As of the Java 2 platform v1.2, Hashtable class was retrofitted to implement the Map interface, making it a member of the Java Collections Framework.

* One of the major differences between HashMap and Hashtable is that HashMap is non-synchronized whereas Hashtable is synchronized, which means Hashtable is thread-safe and can be shared between multiple threads but HashMap cannot be shared between multiple threads without proper synchronization. Java 5 introduced ConcurrentHashMap which is an alternative of Hashtable and provides better scalability than Hashtable in Java.Synchronized means only one thread can modify a hash table at one point of time. Basically, it means that any thread before performing an update on a hashtable will have to acquire a lock on the object while others will wait for lock to be released.
* The HashMap class is roughly equivalent to Hashtable, except that it permits nulls. (HashMap allows null values as key and value whereas Hashtable doesn’t allow nulls).
* The third significant difference between HashMap vs Hashtable is that Iterator in the HashMap is a fail-fast iterator while the enumerator for the Hashtable is not and throw ConcurrentModificationException if any other Thread modifies the map structurally by adding or removing any element except Iterator’s own remove() method. But this is not a guaranteed behavior and will be done by JVM on best effort. This is also an important difference between Enumeration and Iterator in Java.
* One more notable difference between Hashtable and HashMap is that because of thread-safety and synchronization Hashtable is much slower than HashMap if used in Single threaded environment. So if you don’t need synchronization and HashMap is only used by one thread, it out perform Hashtable in Java.
* HashMap does not guarantee that the order of the map will remain constant over time.

Note that HashMap can be synchronized by

**Map m = Collections.synchronizedMap(hashMap);**