**Index**

**---------------------------------------------------------String---------------------------------------**

**Why String is immutable?**

Ans : String is a final class. It implements serializable, charsequence and comparable interface. It is immutable because of following reason:

1. **String Pool facility**: In String Pool one string literal can be reference by more than one reference variable. So if any one of them changes the value of others will be automatically get affected. it is not possible without immutability.

2. **Security:** String is used as parameter in most of the java classes such as host name, port, connection URL. In case if string is not immutable it leads serious security thread because any one can access file which he has authorization and can change the file name and gain access those files. But due to immutability we no need to worry about those kinds of security threads.

3. **Thread safety**: Since String is immutable; it can be safely shared between many threads, which is very important for multithreading programming and to avoid any synchronization issue in java? Immutability makes string thread safe we do not need to synchronize string operation externally.

4. **Immutability allow String to cache its hash code** : Since String is immutable , Immutable string in java caches its has code , and do not calculate every time we call hash code method of String, which makes it very fast as hash map key to be used in hash map in Java.

**Q. How substring works in java?**

Ans: Substring method is defined in java.lang.string class. it is used to get parts of string. It is a overloaded method in java.(substring(int sIndex,eIndex),substring(int sIndex).When sIndex is equal to the length of string It does not through exception ,it return empty string. When sIndex equal to sIndex it throws StringOutOfBoundException.

It calls String (int offset, int count, char value []) constructor to create new String object. The array value[], which is the same character array used to represent original string

This stops original string to be garbage collected, in case if doesn't have any live reference. This is clear case of memory leak in Java, where memory is retained even if it's not required

**Q.What is the difference b/w String, String Buffer and String Builder?**

Ans:

1) String is immutable while String Buffer and String Builder is mutable object.

2) Stringbuffer is synchronized while String Builder is not which makes String Builder faster than Stringbuffer.

3) Concatenation operator "+" is internal implemented using either Stringbuffer or StringBuilder.

4) Use String if you require immutability, use Stringbuffer in java if you need mutable + thread-safety and use StringBuilder in Java if you require mutable + without thread-safety.

**Q: What is the difference b/w comparator and comparable interface and how to sort collection more than one key ?**

Ans : Comparators and comparable in Java are two interfaces which is used to implement sorting in Java.

1. Comparator interface is defined in java.utill package and comparable interface is defined in java.lang

2. Comparator interface has public int compare(Object1 1 , Object2) method return negative,possitive and zero as first args is less than,equal and greater than second object.

Comparable interface has public int compareTo(Object) methods return negative , positive and zero as this object less than equal and greater than specified object.

3. Comparator interface compare two object provided it while comparable interface compare this reference to specified object.

4. Compareable interface is used to implement natural odering of object. If any class implementing comparable interface the collection that object can be sort by Collections.sor and Arrays.sort.

- To compare collection more than one key , we used java 8 comprator.thenComparing method;

5. In Comparable sorting logic must be in same class whose objects are being sorted. Hence this is called natural ordering of objects Sorting logic is in separate class. Hence we can write different sorting based on different attributes of objects to be sorted

**Q.  How to implement equal () and hash code() in java and common error in implementation  ?**

Ans: Equal method should be :

1) **Reflexive:** Object must be equal to itself.

2) **Symmetric:** if a.equals(b) is true then b.equals(a) must be true.

3) **Transitive:** if a.equals(b) is true and b.equals(c) is true then c.equals(a) must be true.

4) **Consistent:** multiple invocation of equals() method must result same value until any of properties are modified. So if two objects are equals in Java they will remain equals until any of there property is modified.

5) **Null comparison:** comparing any object to null must be false and should not result in NullPointerException.

Equals and hashCode contract in Java :

1) If two objects are equal by equals () method then there hashcode must be same.

2) If two objects are not equal by equals () method then there hashcode could be same or different.

**Note:** instead of instanceof we can use getClass() method for type identification because instanceof check returns true for subclass also.

Common Error:

* Instead of overriding equals() method programmer overloaded it.
* Not doing null check for member variables
* Not overriding hashCode method in Java and only overriding equals() method**.**

**Q. How to convert Date in String?**

Ans:

1) First step is to create a date format using SimpleDateFormat class.

2) Call format () method of SimpleDateFormat by passing Date object this will return String representation of date into specified date format.

Date dateNow = **new** Date();

SimpleDateFormat dateformatyyyyMMdd = **new** SimpleDateFormat("yyyyMMdd");

String date\_to\_string = dateformatyyyyMMdd.format(dateNow);

**Q.What does intern () method do?**

**Ans:** When we invoke the intern method if pool contains same string in pool, determined by equal method, the string from pool is returned. Otherwise, this string is added to pool and reference to this string object is returned.

---------------------------------------------------------------------------------------------------------------------

**Q. Difference b/w overloading and overriding?**

Ans:

* We can overload methods within class while overriding can be done within subclass and superclass.
* In case of method overloading in Java, Signature of method changes while in case of method overriding it remain same.
* Cannot override static, final and private method but overload static, final and private.
* Overriden method can throw check exception declared by overridden method or by its subclass and can throw runtimeException.Overloaded method in java doesn’t have such restriction.
* **Overloading** with object auto boxing and var-args:

- In overloading widening beat to boxing and var-args

- In overloading boxing beat var-args

1. Primitive Widening > Boxing > Varargs.   
2. Widening and Boxing (WB) not allowed.   
3. Boxing and Widening (BW) allowed.   
4. While overloading, Widening + vararg and Boxing + vararg can only be used in a mutually exclusive manner i.e. not together.   
5. Widening between wrapper classes not allowed

**Overriding**:

* 1. Final method cannot be override.
* Access modifier of overriding method cannot have more restrict modifier than Overridden method.
* Overriding method cannot throw checked exception, which is not declared by overridden method.
* Overriding method can throw un-check exception, which is not declared by overridden method.
* Overriding method and overridden method return type should be same.
* Sub class cannot have as method with same name as superclass method name but return type is different.
* How to call super class version of method in subclass
* static method cannot be override return type can be subtype or same but can't we wrapper class
* Access modifier can't we more restrictive

------------------------------ **Class, Abstract and interface**--------------------------------------------

**Q.  Difference b/w interface abstract class and where should use them?**

Ans.

* Interface in Java can only contain declaration. You can not declare any concrete methods inside interface. On the other hand abstract class may contain both abstract and concrete methods, which makes abstract class an ideal place to provide common or default functionality.
* Java interface can extend multiple interface also Java class can implement multiple interfaces, Which means interface can provide more polymorphism support than abstract class.
* In order to implement interface in Java, until our class is abstract, we need to provide implementation of all methods,. On the other hand abstract class may help us in this case by providing default implementation.
* Interface is declared with interface keyword while abstract class is declared with abstract keyword.
* Interface can contain only constant while abstract class can contain both constant and non constant.
* Abstract class is slightly faster than interface because interface involves a search before calling any [overridden method in Java](http://java67.blogspot.sg/2012/08/what-is-method-overriding-in-java-example-tutorial.html).
* Interface is better suited for Type declaration and abstract class is more suited for code reuse and evolution perspective.

**When to use interface and abstract class in Java :**

* If we want to implement multiple inheritance than we should use interface instead of abstract class.
* If we want to some common functionality than abstract class is better than interface.
* Interface also provide more decoupling than abstract class because interface doesn't contain any implementation detail, while abstract class may contain default implementation which may couple them with other class or resource.

**Q. Can be instantiate the interface?**

* Ans : instantiate an interface with [anonymous class](http://www.journaldev.com/996/java-nested-classes-java-inner-class-static-nested-class-local-inner-class-and-anonymous-inner-class).

# Q. why we need to override hash code and equal ()?

# Ans : To store unique object in collection or data structure we need to override hascode and equal methods and to get object from map we have to override hascode. Q.  Can a abstract class have a constructor in java if yes or no why?

Ans : Yes a abstract class can have a constructor.

**Q: What is Static Final blank variable?**

Ans: A static final variable is not initialized at the time of declaration is known as static final blank variable. Static final variable only initialized in static block.

**Q. Use of this and super keyword in java?**

Ans :

This:

* This keyword is used to current class instance variable.
* This keyword is used to current class method.
* This keyword is used to current class constructor.
* This keyword can be passed as parameter in method.
* This keyword can be passed as argument in constructor.
* This keyword is used to return current class object.

Super:

* Super keyword used to call base class constructor.
* Super keyword is used to call base class method.
* Super keyword is used to access base class instance member.

**Q. What is marker or tagged interface?**

Ans : An interface that doesn’t have any data member is known as marker interface.

**Q. What is Nested Interface?**

Ans: An interface has inside another interface known as nested interface.

**Q. What will the result for below code ?**

Dog d=null;

d.instanceof Dog

**Q. Can Constructor have private modifier?**

Ans : yes, but we can not create instance of that class.

------------------------------------ **Exception Handling:**-----------------------------------------------------

**Q. What is the difference b/w throw and throws?**

Ans.

* We can declare multiple exceptions thrown by a method in throws separating them in comma while we can throw only one instance of exception in throw.
* We can not use throws inside static and switch statement while throw can use.

Q. **What is the checked and unchecked exception?**

Ans :

* Checked exceptions are the direct subclass of Exception. While unchecked exceptions are the direct subclass of Runtime Exception.
* Checked exception is required handle at compile time while unchecked exception not required.

----------------------------------------------------------------------------------------

File Handling

-------------------------------------------------------------------------------------

Thread  
**Q. what is the difference b/w sleep ad wait?**

Ans.

* wait is called from synchronized context only while sleep can be called without synchronized block
* wait is called on Object while sleep is called on Thread
* Wait release lock on object while waiting while sleep doesn’t release lock while waiting.
* Waiting thread can be awake by calling notify and notifyAll while sleeping thread can not be awaken by calling notify method.
* Wait is normally done on condition, Thread wait until a condition is true while sleep is just to put your thread on sleep.

**Q what is the difference b/w sleep and yield?**

Ans:

* Thread. Yield () method pause the currently executing thread temporarily for giving chance to other waiting thread of same priority to execute. if there is not waiting thread or all waiting threads are lower priority then same thread will continue its execution.
* Sleep methods put in sleep **currently executing thread** for the specified number of milliseconds plus the specified number of nanoseconds.

**Q what is the difference b/w Thread and Runnable ?**

* Thread is a class while runnable is a interface.
* Inheriting all Thread methods are additional overhead just for representing a Task, which can be done easily with Runnable.
* LooseCoupling,functional overhead,reuseability,inheritance option

**Q. Life cycle of Thread?**

- New --🡪ready -🡪runnable 🡪 blocking 🡪dead

**Q Difference start and run method ?**

Ans :

* When program calls start() method a **new Thread** is created and code inside run() method is executed in new Thread while if you call run() method directly **no new Thread is created** and code inside run() will execute on **current Thread**.
* **Can not call start() method twice** on thread object. once started, second call of start() will throw IllegalStateException in Java while you can call run() method twice.

**Q. How volatile keyword works in java?**

Ans :

* Volatile keyword is used as indicator for java compiler or thread that do not cache the value of this variable always read from main memory.
* If we want to share a variable in which read and write operation is atomic by implementation we can declare them as volatile.

--------------------------------------------------------------------------------------------------

**Q. What is the exception?**

Ans : A exception is a unexpected result.

**Q. What is the runtime or unchecked and check exception?**

Ans :

**Unchecked exception :**

* This type of exception occurs due to programming problem.
* These types of exceptions are the direct subclass of runtime Exception.

**Check Exception**

* Checked exception are the exceptions which forces the programmer to catch them explicitly in try-catch block.
* It is a subclass of Exception

**Q. What is the difference b/w exception and error?**

Ans :

* An error is an irrecoverable condition occurring at runtime.
* While exceptions are conditions that occur because of bad input etc.

Q. **What is difference in final, finalize and finally keyword in Java?**

Ans :

* Final keyword used to create immutable method or class. By making a class final we cannot extend the class, same we cannot override a final method.
* Finalize method called by garbage collector to clean object before it is collected.
* Finally keyword is used in error or exception handling.

**Q. What is the difference b/w ant and maven?**

Ans:

* Ants do not come with formal conventions such as a common project directory. Maven consists of conventions.
* While Ant is procedural, Maven is declarative
* Ant does not have a life cycle whereas Maven has a life cycle.
* The scripts in Ant are not reusable whereas Maven comes with reusable plugins.

**Q. Compare different version of java ?**

Ans :

**Java 5 :**

* Generics
* Enhanced for Loop
* Autoboxing/Unboxing
* Typesafe Enums
* Varargs

**Java 6 :**

* Scripting Language Support
* Pluggable Annotations
* Native PKI, Java GSS, Kerberos and LDAP support.
* Integrated Web Services.

**Java 7:**

* Binary literal
* String in switch statement
* Try with resource
* Catching multiple exception in single catch block
* Underscore in numeric literal
* Type inference generic instance creation
* Map<String, List<String>> myMap = new HashMap<String, List<String>>();
* Map<String, List<String>> myMap = new HashMap()

Java 8 :

* [forEach() method in Iterable interface](http://www.journaldev.com/2389/java-8-features-for-developers-lambdas-functional-interface-stream-and-time-api#iterable-forEach)
* [default and static methods in Interfaces](http://www.journaldev.com/2389/java-8-features-for-developers-lambdas-functional-interface-stream-and-time-api#interface-default-static-method)
* [Functional Interfaces and Lambda Expressions](http://www.journaldev.com/2389/java-8-features-for-developers-lambdas-functional-interface-stream-and-time-api#functional-interface-lambdas)
* [Java Stream API for Bulk Data Operations on Collections](http://www.journaldev.com/2389/java-8-features-for-developers-lambdas-functional-interface-stream-and-time-api#java-stream-api)
* [Java Time API](http://www.journaldev.com/2389/java-8-features-for-developers-lambdas-functional-interface-stream-and-time-api#java8-time)
* [Collection API improvements](http://www.journaldev.com/2389/java-8-features-for-developers-lambdas-functional-interface-stream-and-time-api#java8-collection)
* [Concurrency API improvements](http://www.journaldev.com/2389/java-8-features-for-developers-lambdas-functional-interface-stream-and-time-api#java8-concurrency)
* [Java IO improvements](http://www.journaldev.com/2389/java-8-features-for-developers-lambdas-functional-interface-stream-and-time-api#java8-io)
* [Miscellaneous Core API improvements](http://www.journaldev.com/2389/java-8-features-for-developers-lambdas-functional-interface-stream-and-time-api#java8-core)

http://www.journaldev.com/2389/java-8-features-for-developers-lambdas-functional-interface-stream-and-time-api

**Q Difference b/w web server an application server?**

Ans

* Application Serversupports distributed transaction and EJB. While Web Server only supports Servlets and JSP.
* Application server support different type of request, web server support only http request.

------------------------------------------------------------------------------------------------------

Enum :

**Q1. Can Enum implement interface and extends class in Java?**

Ans : Yes, Enum can implement interface in Java. Since enum is a type, similar to class and interface, it can implements interface but can not extends class.

**Q2. Can Enum extends class in Java?(** **http://javarevisited.blogspot.in/2011/08/enum-in-java-example-tutorial.html)**

Ans : No, Enum can not extend class in Java. Since all Enum by default extend abstract base class java.lang.Enum, obviously they can not extend another class, because Java doesn't support multiple inheritance .

**Q3. How do you create Enum without any instance? Is it possible without compile time error?**

Ans : Yes

**Q4. Can we override toString() method for Enum? What happens if we don't?**

Ans : yes

Q5. **Can we create instance of Enum outside of Enum itself? If Not, Why?**

Ans: No, No, you can not create enum instances outside of Enum boundry, because Enum doesn't have any [public constructor](http://java67.blogspot.sg/2012/12/how-constructor-chaining-works-in-java.html), and compiler doesn't allow you to provide any public constructor in Enum.

**Q6. Can we declare Constructor inside Enum in Java?**

Ans . Yes, you can, but remember you can only declare either private or package-private constructor inside enum.

**Q7. Can we use Enum with TreeSet or TreeMap in Java?**

Ans . Since Enum by default impalement Comparable interface, they can be safely used inside TreeSet or TreeMap in Java.

**Q8. How to convert an String to Enum and enum to string in Java?**

Ans : By using valueOf() we convert enum to string and name() convert string to enum.

Q. **Shallow cloning and deep cloning?**

Ans : 1.

* In case of shallow cloning we use default implementation of clone method while in case of deep cloning we provide our own implementation.
* In shallow copy main or parent object is copied, but they share same fields or children if fields are modified in one parent object other parent fields have automatic same changes occur, but in deep copy this is not the case.
* Deep copy is expensive as compare to shallow copy in terms of object creation, because it involves recursive copying of data from other mutable objects, which is part of original object.
* If our parent object contains only primitive value then shallow copy is good for making clone of any object because in new object value is copied but if parent
* object contains any other object then only reference value is copied in new parent object and both will point to same object so in that case according to our need we can go for deep copy.

---------------------------------------------------------------------------------------------

Inner class

Q. **What are Static Nested classes?**

* If the nested class is static, then it’s called static nested class.
* Static nested classes can access only static members of the outer class.
* OuterClass.StaticNestedClass nestedObject =new OuterClass.StaticNestedClass();

Q. **What are the Inner classes?**

**Ans :**

**Inner class**:

* Any non-static nested class is known as inner class.
* Inner classes are associated with the object of the class and they can access all the variables and methods of the outer class.
* Since inner classes are associated with instance, we can’t have any static variables in them.
* Object of inner class are part of the outer class object and to create an instance of inner class, we first need to create instance of outer class.

|  |  |
| --- | --- |
|  | OuterClass outerObject = new OuterClass();  OuterClass.InnerClass innerObject = outerObject.new InnerClass(); |

Q **What are the local inner classes?**

* If a class is defined in a method body, it’s known as local inner class.
* Since local inner class is not associated with Object, we can’t use private, public or protected access modifiers with it. The only allowed modifiers are abstract or final.
* A local inner class can access all the members of the enclosing class and local final variables in the scope it’s defined.
* InnerClass obj = new InnerClass()

**Anonymous inner class**:

* A local inner class without name is known as anonymous inner class.
* An anonymous class is defined and instantiated in a single statement.
* Anonymous inner class always extend a class or implement an interface.
* Since an anonymous class has no name, it is not possible to define a constructor for an anonymous class.
* Anonymous inner classes are accessible only at the point where it is defined.

Advantage :

* Readable and maintenance
* More encapsulation

Q**. Static and transient variable can be serialized or not?**

* NO

**Q. How serialVersionUID generated?**

- Class variable, method,classname and package

**Q.Is sub class serializable if super class implements serializable interface.**

**Ans.** Yes, subclass automatically serialized if super class is serializable.

**Q Serializable with aggregation, collection, array?**

**Ans.** If a class having instance of another class then the other class must be serializable, to serlized the main class.

**Q. Static and transient variable can be serialized or not?**

Ans. No

Q **How system.out.println works?**

Ans : system is a class which is having the out as static variable of printstream type.

PrintStream is a clas which is having println overlaoded method .

JVM initiallize the system class by calling initializesystem()

Q **Why do we need overloading and overriding?**

**Q why main method is public ,void and static ?**

Ans : Main method is public so that JVM can access outside of class .

- it is void beacuse it is just enty point to program and returing any things.

- it is static so that JVM can access without creating the instance of class.

Q **Can static method overide ?**

Ans : No it will give compile time error.

Q **Can abstract method can have static method and type abstract ?**

Ans : Yes but can be abstract.

Q **Willl final blick run after return?**

Ans : yes

Q **Can constructors be synchronized in Java?**

Ans : No

**Q Does Java pass by reference or by value?**

Ans : java is pass by value because jmv create the copy of value and pass those value as argument to method.

Q .what is interface collision.

Ans.

**Q. How to create immutable class in java and benefits.**

Ans.

1) Don’t provide “setter” methods — methods that modify fields or objects referred to by fields.

2) Make all fields final and private

3) Don’t allow subclasses to override methods

4) Special attention when having mutable instance variables

Benefits :

1. are simple to construct, test, and use
2. are automatically thread-safe and have no synchronization issues
3. do not need a copy constructor
4. do not need an implementation of clone
5. allow [hashCode](http://howtodoinjava.com/2012/10/09/working-with-hashcode-and-equals-methods-in-java/) to use lazy initialization, and to cache its return value
6. do not need to be copied defensively when used as a field
7. make good [Map keys and Set elements](http://howtodoinjava.com/2012/10/09/how-hashmap-works-in-java/) (these objects must not change state while in the collection)
8. have their class invariant established once upon construction, and it never needs to be checked again
9. always have “**failure atomicity**” (a term used by Joshua Bloch) : if an immutable object throws an exception, it’s never left in an undesirable or indeterminate state

**Q . Composition vs inheritance ?**

**Ans :**

**1. Need to change subclass implementation:** suppose super class implementation is not our hand. Super class implementation change then we need to change subclass implementation to solve compile time error.

2. **Security**:

Another problem with inheritance is that we are exposing all the super class methods to the client and if our super class is not properly designed and there are security holes, then even though we take complete care in implementing our class, the poor implementation of super class affects us.  
Composition helps us in providing controlled access to the super class methods whereas inheritance does not provide any control of the super class methods,

3. Another benefit with composition is that it provides flexibility in invocation of methods.

**Unit testing is easy**

1. Unit testing is easy in composition because we know what all methods we are using from super class and we can mock it up for testing whereas in inheritance we depend heavily on super class and don’t know what all methods of super class will be used.

**Aggregation vs Composition**

1. Dependency: Aggregation implies a relationship where the child can exist independently of the parent. For example, Bank and Employee, delete the Bank and the Employee still exist. Whereas Composition implies a relationship where the child cannot exist independent of the parent. Example: Human and heart, heart don’t exist separate to a Human
2. Type of Relationship: Aggregation relation is “has-a” and composition is “part-of” relation.
3. Type of association: Composition is a strong Association whereas Aggregation is a weak Association.

**Q. Auto boxing, un-boxing, and overhead**

- Auto Boxing: primitive to wrapper conversion

- Un-Boxing: Wrapper to primitive conversion

In method overloading, boxing and unboxing can be performed. There are some rules for method overloading with boxing:

Widening beats boxing

Widening beats varargs

Boxing beats varargs

Q. ***Why constructor are not static?***

- Constructor are not static because it violate the inheritance principle.

it constructor are static then sub class cannot access super class constructor.

***Q. Why need marker interface?***

- Marker interface are indication to JVM do some operation

- Thread safe

- Code review tool easily can find bugs

***Q. What happened if we print java object or default implementation of toString() in Object class?***

- It prints getClass().getName() + '@' + Integer.toHexString(hashCode())

Q. **Can we serialize static method and class?**

**-** No we cannot, because we serialized object state, static method and variable belongs to class not object.

.

<http://beginnersbook.com/2013/12/difference-between-arraylist-and-hashmap-in-java/>

<http://www.tutorialspoint.com/listtutorials/java/collections/1>

http://www.jusfortechies.com/java/core-java/deepcopy\_and\_shallowcopy.php