1. **Which two method you need to implement for key Object in Hash Map?**

**Equals and Hashcode** are the two methods we need to implement for key object in the Hash Map. If we doesn’t override the equals and hashcode then the jvm doesn’t understand on what basis the key objects should be compared. If we doesn’t override those two methods and if we try to insert the two objects which are having the same values, it will insert them because the jvm doesn’t know that both the objects are same since the two same objects has the different hashcode. Since their hashcode is different there is no point that the objects will be equal. So we should override the equals and hashcode so that the jvm doesn’t allow the duplicate keys. And if the same key already exists then value will be overridden by the new value.

1. **How Hash Map works in Java?**

Hash Map works on the principle of hashing. There are three terms in hashing.

* Hash Function
* Hash Value
* Bucket

Here Hash function is nothing but the hashcode() method. It returns the integer value. This value is the hash value. This value is used to identify the buckets.

Bucket is nothing but the list that contain the key-value pairs. It uses simple linked list to store the key-value pairs.

Whenever the get(Key) method is called, it is going to call the hashcode() method of the object to get the hash value. Based on the hash value it finds the right bucket of the object. If the bucket contains more than one entry (Key-value pairs), then it is going to call the equals() method to identify the correct key, and it gets the value for the identified key.

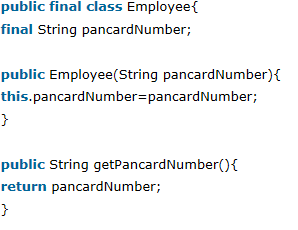
If the key is null, then it matches the index to 0 which doesn’t contain any elements.

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

Immutable object is an object which cannot be changed or modified once it is created. Any modification in the immutable objects results in the creation of the new object.

String class and wrapper classes are examples of the immutable object.

We can make the object as immutable by making the class as final and variables as private and final, and we can modify the value only in the constructor and no setter methods.



1. **What is the difference between creating the string using new and literal?**

When we create string using the new operator it is created in the heap not in the string constant pool. Whereas if we use literal it is created in the string constant pool.

If we want the object that is created using the new operator in the string constant pool then we use intern() method to place the object in the string pool.

1. **What is the difference between the String buffer and String Builder?**

StringBuffer is synchronized and its thread safe.

StringBuilder is not synchronized and its not thread safe.

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

|  |  |
| --- | --- |
| **ArrayList** | **Vector** |
| Not Synchronized | Synchronized |
| Performance is fast | Performance is slow as it is thread safe |
| Only Iterator is used for traversal | Enumeration and Iterator can be used |
| Cannot define increment size | Defines increment size |

1. **Define Design Pattern?**

Design Pattern is a solution to the commonly occurring problems in the software design.

1. **What is the difference between factory and Abstract Factory Design pattern?**

With Factory interface you produce implementations(Apple, Banana etc) of a particular interface - IFruit

With Abstract factory you produce implementations of few factory interface - IFruitFactory

1. **What is a singleton? Is it better to make the whole method synchronized or critical section synchronized?**

Singleton in java is a class with only one instance. We can create the singleton class by making all the data members static.

There are two forms of singleton design pattern

* Lazy Instantiation
* Early Instantiation

Lazy Instantiation – Create an instance when required

Early Instantiation – Crate an instance at the time of loading

Main advantage of singleton class is that it creates only only object so that memory can be saved.

Mainly singleton is used in case of multi-threaded environment and database applications.

It is used for logging, caching, thread pools, configuration etc.

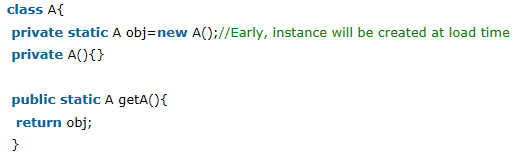
Create a singleton design pattern:

* Static data member – only one memory it gets
* Private constructor – so that the object cannot be created outside the class
* Static factory method – This returns the instance to the caller.

**Early Instantiation**

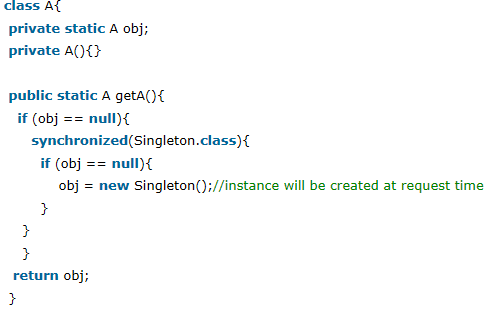
We create the instance of the class at the time of declaring the static data member.

So the instance of the class is created at the time of loading.



**Lazy Instantiation**

We create the instance of the class in the synchronized block so that it is created when required.



**Note:**

* **If singleton class is loaded by two different class loaders then two instances are created, one for each loader.**
* **If the singleton class is serializable, you can serialize the singleton instance. But while you deserialize it doesn’t return anything as the static and transient variables cannot be serialized.**
* **To resolve this issue you need to override the readResolve() that enforces the singleton object. This method should be called after the deserialization so that it returns the singleton object.**
* **There is also a design pattern MONOSTATE which acts just like singleton pattern. Here all the data and methods are marked static.**

1. **Is it better to synchronize the whole getInstance() or the critical section of the getInstance() method?**

It is better to synchronize the part of the critical section of the method because it improves the performance compared to the marking entire method as synchronized.

If we lock the whole method, if any thread locks the entire method and the other thread wants to access the method has to wait even though creation of the object is not necessary.

1. **When do you override equals and hashcode?**

We override them when we need to check the equality of two objects and when we want to use the object as a key to hashmap.

1. **What will be the problem if you don’t override the hashcode method?**

If we want to get the value of a particular key in hashmap, then it will not be able to recover that object. It returns null value even if the key is present in the hashmap.

1. **Does not overriding the hashcode() has any performance implications?**

Definitely. Because the poor hashcode will lead to the frequent collision and reduces the performance and increases the time in getting the object.

1. **What happens if you call return statement or system.exit in try or catch blocks? Will finally execute?**

When a return statement is called then the finally is executed without fail.

But if it encounters the system.exit it doesn’t execute the finally block.

1. **Can you override private or static method in java?**

No we cannot because, the private methods will not be visible to the other classes so we can’t override. And coming to static methods, since these are the class level methods we can’t override. If you create the same method with the same signature in this case it is called method hiding.

1. **If a method throws null pointer exception in the super class can the sub class method throw the runtime exception?**

Yes, it can but only in case of runtime exception.

If it is a checked exception we cant do it.

1. **How do you ensure that N threads accessing N resources prevent deadlock?**

If threads access the resources in particular order and release in the reverse order deadlock can be prevented.

1. **Can you access non static variable in the static context?**

No you cannot.

[Static variable in Java](http://javarevisited.blogspot.com/2011/11/static-keyword-method-variable-java.html) belongs to Class and its **value remains same for all instance**. static variable initialized when class is loaded into [JVM](http://javarevisited.blogspot.com/2011/12/jre-jvm-jdk-jit-in-java-programming.html) on the other hand instance variable has different value for each instances and they get created when instance of an object is created either by using new() operator or using reflection like Class.newInstance(). So if you try to access a non static variable without any instance compiler will complain because **those variables are not yet created** and they don't have any existence until an instance is created and they are associated with any instance. So in my opinion only reason which make sense to disallow [non static or instance variable](http://javarevisited.blogspot.com/2012/02/difference-between-instance-class-and.html) inside static context is nonexistence of instance.

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

1. **What is Abstraction?**

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

1. **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.

1. **How many ways we can create an object?**

* New operator
* Clone
* getInstance
* class.forName() – this method internally uses getInstance()
* Deserialization

1. **Serialization**

* Static variables can be serialized if we provide the values to those fields during initialization.

Static int count=10;

* Transient variables cannot be serialized eve if we provide the value during the initialization.
* Transient and static in combination can be serialized since static will govern the behavior of the variable, not the transient.
* If we mark the transient field as final then the field can be serialized.
* Super class fields will not be serialized if the sub class is serializable.
* But the fields that are marked final and static in the super class can be serialized.
* If the class extends the interface, and it consists of the final variables, then the class that is serializable can serialize the field of the interface since all the fields in the interface are final by default.
* <http://javabeginnerstutorial.com/core-java-tutorial/transient-vs-static-variable-java/>

1. Can the fields of the sub class override the fields of the super class?

No, the sub class fields cannot override the super class fields, the object is checked at compile time rather than at compile time.

Even though the base class reference refers to the derived class object then the field of the base class object will be called, since the object is checked at the compile time.