Final Keyword In Java

Java final variable

1. cannot change the value of final variable (It will be constant).
2. It can be initialized only in constructor.
3. A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.

Java final method

1. you cannot override it.
2. final method is inherited but you cannot override it.
3. If you declare any parameter as final, you cannot change the value of it.
4. constructor can't be made final because is never inherited.

Java final class

1. you cannot extend it.

Java static keyword

*The **static keyword** in Java is used for memory management mainly.* ***Java static property is shared to all objects.***

Java static variable

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

Java static method

- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.
- The static method cannot use non static data member or call non-static method directly.
- this and super cannot be used in static context.

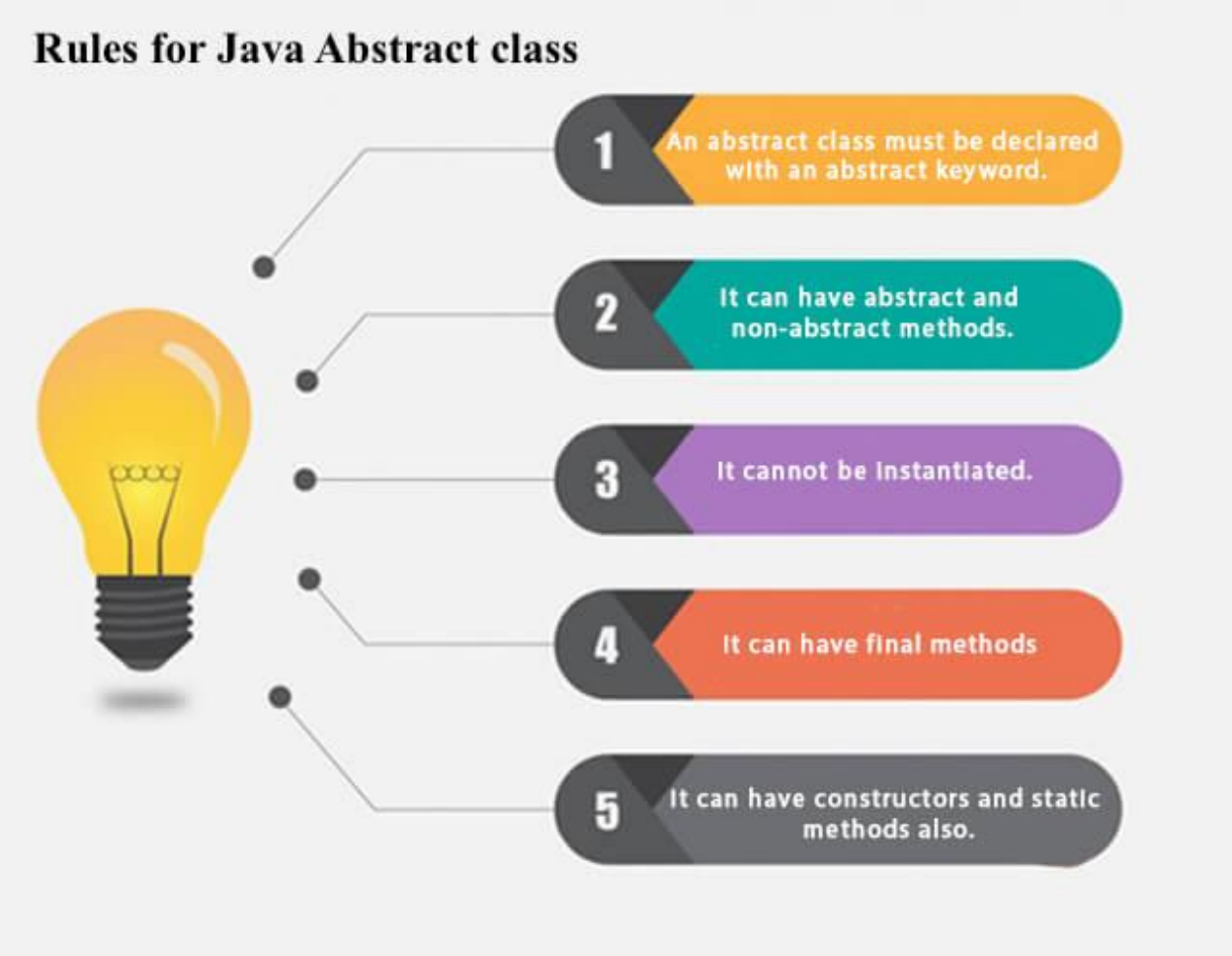
Why is the Java main method static?

It is because the object is not required to call a static method. If it were a non-static method, JVM creates an object first then call main() method that will lead the problem of extra memory allocation.

Java static block

- Is used to initialize the static data member.
- It is executed before the main method at the time of classloading.

Abstract class in Java : Factory Design Pattern



Interface in Java

1. Since Java 8, interface can have default and static methods.
2. An interface which has no member is known as a marker or tagged interface, for example, Serializable, Cloneable, Remote,

> ***The Java compiler adds public and abstract keywords before the interface method. Moreover, it adds public, static and final keywords before data members***

3. Since Java 8, we can have method body in interface. But we need to make it default method
4. we can have static method in interface

```
interface Drawable{
    void draw();
    default void msg(){System.out.println("default method");}
    static int cube(int x){return x*x*x;}
}
```

Nested Interface in Java

```
interface printable{
        void print();
        interface MessagePrintable{
                void msg();
        }
}
```

Access Modifiers in Java

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Object Cloning in Java

The **java.lang.Cloneable interface** must be implemented by the class whose object clone we want to create. If we don't implement Cloneable interface, clone() method generates **CloneNotSupportedException**.

```
protected Object clone() throws CloneNotSupportedException
```

Object.clone() doesn't invoke any constructor so we don't have any control over object construction.

Object.clone() supports only shallow copying but we will need to override it if we need deep cloning.

shallow copying : only primitive data types get copied.

Java Strictfp Keyword

Java strictfp keyword ensures that you will get the same result on every platform if you perform operations in the floating-point variable. The precision may differ from platform to platform that is why java programming language have provided the strictfp keyword, so that you get same result on every platform. So, now you have better control over the floating-point arithmetic.

1. The strictfp keyword can be applied on methods, classes and interfaces.
2. The strictfp keyword **cannot** be applied on abstract methods, variables or constructors.

Java String compare

1. By equals() method : **authentication**
2. By = = operator : **sorting**
3. By compareTo() method : **reference matching**

```
class Teststringcomparison3{
        public static void main(String args[]){
          String s1="Sachin";
          String s2="Sachin";
          String s3=new String("Sachin");
          System.out.println(s1==s2);//true (because both refer to same instance)
          System.out.println(s1==s3);//false(because s3 refers to instance created in non-pool)
        }
        }
```

S1 and s2 points to same string in Pool

S3 refers to new string from non-pool.

Java Inner Classes

Non-static nested classes are known as inner classes.

Type	Description
Member Inner Class	A class created within class and outside method.
Anonymous Inner Class	A class created for implementing interface or extending class. Its name is decided by the java compiler.
Local Inner Class	A class created within method.
Static Nested Class	A static class created within class.
Nested Interface	An interface created within class or interface.

Java Member inner class

A non-static class that is created inside a class but outside a method is called member inner class.

```
class TestMemberOuter1{
 private int data=30;
 class Inner{
  void msg(){System.out.println("data is "+data);}
 }
 public static void main(String args[]){
  TestMemberOuter1 obj=new TestMemberOuter1();
  TestMemberOuter1.Inner in=obj.new Inner();
  in.msg();
 }
}
```

Java Anonymous inner class

```
abstract class Person{
  abstract void eat();
}
class TestAnonymousInner{
 public static void main(String args[]){
  Person p=new Person(){
  void eat(){System.out.println("nice fruits");}
  };
  p.eat();
 }
}
```

Java Local inner class

1. A class i.e. created inside a method is called local inner class in java.

> 1. ***Local inner class cannot be invoked from outside the method.***
> 2. ***access the non-final local variable in local inner class.***

Java static nested class

- It can access static data members of outer class including private.
- Static nested class cannot access non-static (instance) data member or method.

```
class TestOuter1{
  static int data=30;
  static class Inner{
    void msg(){System.out.println("data is "+data);}
  }
  public static void main(String args[]){
  TestOuter1.Inner obj=new TestOuter1.Inner();
  obj.msg();
  }
}
```

Java Nested Interface

- Nested interface must be public if it is declared inside the interface

```
interface interface_name{
 ...
 interface nested_interface_name{
  ...
 }
}
```

- it can have any access modifier if declared within the class.

```
class class_name{
 ...
 interface nested_interface_name{
  ...
 }
}
```

- Nested interfaces are declared static implicitely.

```
interface Showable{
  void show();
  interface Message{
    void msg();
  }
}
class TestNestedInterface1 implements Showable.Message{
 public void msg(){System.out.println("Hello nested interface");}

 public static void main(String args[]){
  Showable.Message message=new TestNestedInterface1();//upcasting here
  message.msg();
 }
}
```

Can we define a class inside the interface?

Yes, If we define a class inside the interface, java compiler creates a static nested class.

```
interface M{
  class A{}
}
```

HashMap working and implementations

1. It uses hashing and chaining.
2. Hashing to find the bucket number using hashCode() OR hash function
3. Chaining to append the item in the same bucket list having same hashcode

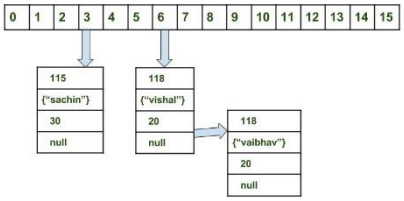
---

4. **equals():** It checks the equality of two objects. It can be overridden. If you override the equals() method, then it is mandatory to override the hashCode() method.
5. **hashCode():** It returns the memory reference of the object in integer form. The value received from the method is used as the bucket number. The bucket number is the address of the element inside the map**. Hash code of null Key is 0**.**Hashfucntion could be anything, for example**

For a string as key , `@Override`

```
public int hashCode()
{
   return (int)key.charAt(0);//use first char ASCII
}
```

6. **Buckets:** Array of the node is called buckets. Each node has a data structure like a LinkedList. More than one node can share the same bucket. It may be different in capacity.
    a. **Indexing to bucket ,Can use % function, for example hascode(key)%(size of map)**



In java 8, the hash will change from using a linked list to a balanced tree, which will improve the worst case performance from O(n) to O(log n).

https://www.geeksforgeeks.org/internal-working-of-hashmap-java/

Difference between array and array list, LinkedList, when to choose which

Array – Direct insertion, O(1) add, delete, search.

Array list – uses dynamic array, so insertion takes shifting, search and delete is O(1)

Linked list – uses doubly list, insertion is simple append so O(1), search and delete O(n);

Diffenece between hashmap and treemap and when to choose which

hashMap uses LinkedList and Balanced Tree, while Treemap uses Red-Black Tree.

Hashmap no sorting, Treemap sorting.

Hashmap O(1) in insert, delete, search best case, while treemap(log n) always

---

What is the difference between string, string buffer and string builder?

Thread safe means – No 2 threads can call same methods of a instance, simultaneously

StringBuilder is faster and preferred over StringBuffer for single threaded program

Index	*String*	*String Buffer*	*String Builder*
Storage Area	Constant String Pool	Heap	Heap
Modifiable	No (immutable	Yes(mutable)	Yes(mutable)
Thread Safe	Yes	Yes	No
Thread Safe	Fast	Very slow	Fast

Final variable in Java cannot be changed. Once if **we** have assigned the **final variable** it **can** not be changed it is fixed. but if you have declare a blank **final variable** then you **can** assign value to it only in constructor.

How to create Immutable class in Java?

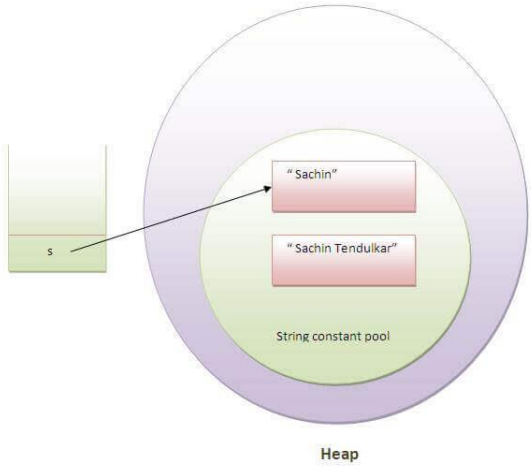
Immutable class means that once an object is created, we cannot change its content. In Java, all the wrapper classes (like Integer, Boolean, Byte, Short) and String class is immutable. We can create our own immutable class as well.
Following are the requirements:

- The class must be declared as final (So that child classes can't be created)
- Data members in the class must be declared as final (So that we can't change the value of it after object creation)
- A parameterized constructor
- Getter method for all the variables in it
- No setters(To not have the option to change the value of the instance variable)

Why string objects are immutable in java?

Because java uses the concept of string literal. Suppose there are 5 reference variables, all referes to one object "sachin". If one reference variable changes the value of the object, it will be affected to all the reference variables. That is why string objects are immutable in java.

```
class Testimmutablestring{
 public static void main(String args[]){
   String s="Sachin";
   s.concat(" Tendulkar");//concat() method appends the string at the end
   System.out.println(s);//will print Sachin because strings are immutable objects
 } }
```




```
}
```

Comparable	Comparator
1) Comparable provides a **single sorting sequence**. In other words, we can sort the collection on the basis of a single element such as id, name, and price.	The Comparator provides **multiple sorting sequences**. In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc.
2) Comparable **affects the original class**, i.e., the actual class is modified.	Comparator **doesn't affect the original class**, i.e., the actual class is not modified.

3) Comparable provides **compareTo() method** to sort elements.	Comparator provides **compare() method** to sort elements.
4) Comparable is present in **java.lang** package.	A Comparator is present in the **java.util** package.
5) We can sort the list elements of Comparable type by **Collections.sort(List)** method.	We can sort the list elements of Comparator type by **Collections.sort(List, Comparator)** method.

Comparable Vs Comparator

To summarize, if sorting of objects needs to be based on natural order then use Comparable whereas if sorting needs to be done on attributes of different objects, then use Comparator in Java.

Covariant Return Type

The covariant return type specifies that the return type may vary in the same direction as the subclass

---

```
class A{
        A get(){return this;}
}

class B1 extends A{
        B1 get(){return this;}
        void message(){System.out.println("welcome to covariant return type");
}

public static void main(String args[]){
        new B1().get().message();
        }
}
```

Overloading and overriding

No.	Method Overloading	Method Overriding
3)	*parameter must be different*.	*parameter must be same*.
4)	Method overloading is the example of *compile time polymorphism*.	Method overriding is the example of *run time polymorphism*.
5)	*Return type can be same or different* in method	*Return type must be same or covariant(open security)* in method overriding.

Exceptions

A ***segmentation fault*** occurs when a program attempts to access a memory location that it is not allowed to access, or attempts to access a memory location in a way that is not allowed.
