# JAVA FAQ

```
class Test
 {
   protected int x, y;
 }
class Main
{
    public static void main(String args[])
 {
        Test t = new Test();
        System.out.println(t.x + " " + t.y);
 }
 }
```

- Output: *0 0*
- In Java, a protected member is accessible in all classes of same package and in inherited classes of other packages.
- Since Test and Main are in same package, no access related problem in the above program.
- Also, the default constructors initialize integral variables as 0 in Java

```
class Test
 {
   public static void main(String[] args)
   {
      for(int i = 0; 1; i++)
     {
         System.out.println("Hello");
         break;
      }
   }
}
```

- Output: Compiler Error
- There is an error in condition check expression of for loop. Java differs from C++(or C) here.
- C++ considers all non-zero values as true and 0 as false. Unlike C++, an integer value expression cannot be placed where a boolean is expected in Java.
- We can rewrite the program by replacing 1 in for loop with true.
- Output: Hello

```
class Main {
   public static void main(String args[])
 {
      System.out.println(fun());
   }
   int fun()
   {
      return 20;
   }
}
```

- Output: Compiler Error
- Like C++, in Java, non-static methods cannot be called in a static method.
- If we make fun() static, then the program compiles fine without any compiler error.
- Output: 20

```
class Test
 {
   public static void main(String args[])
  {
      System.out.println(fun());
   }
   static int fun()
  {
      static int x= 0;
      return ++x;
   }
}
```

- Output: Compiler Error
- Unlike C++, static local variables are not allowed in Java.
- We can declare the x as static int before the main method.
- Output: 1

```
class Point {
   protected int x, y;
     public Point(int _x, int _y) {      x = _x;      y = _y;
   }
}

public class Main {
   public static void main(String args[]) {
    Point p = new Point();
    System.out.println("x = " + p.x + ", y = " + p.y);
   }
}
```

- Output: Compiler Error
- In the above program, there are no access permission issues because the Test and Main are in same package and protected members of a class can be accessed in other classes of same package.
- The problem with the code is: there is not default constructor in Point.
- Remove the parametrized constructor
- Output: x = 0, y = 0

```
class Base {
    void foo() {System.out.println("Base");}
}
class Derived extends Base {
   Protected void foo() {System.out.println("Derived")}
}
public class Main {
   public static void main(String args[]) {
      Derived d = new Derived();
      d.foo();
   }
}
```

- Output: Compiler Error
- foo() is protected in Base and default in Derived. Default access is more restrictive.
- When a derived class overrides a base class function, more restrictive access can’t be given to the overridden function.
- If we make foo() public in derived, then the program works fine without any error.
- Output: Derived

```
public class Main
{
   public static void gfg(String s)
   {
      System.out.println("String");
   }
   public static void gfg(Object o)
   {
      System.out.println("Object");
   }
    public static void main(String args[])
   {
      gfg(null);
   }
 }
```

- Output: String
- Explanation : In case of method overloading, the most specific method is chosen at compile time.
- As ‘java.lang.String’ is a more specific type than ‘java.lang.Object’.
- In this case the method which takes ‘String’ as a parameter is choosen.

```
public class Main
{
   public static void gfg(String s)
   {
      System.out.println("String");
   }
   public static void gfg(Object o)
   {
      System.out.println("Object");
   }
   public static void gfg(Integer i)
   {
      System.out.println("Integer");
   }
     public static void main(String args[])
   {
      gfg(null);
   } }
```

- Output: Compiler Error
- Explanation: In this case of method Overloading, the most specific method is choosen at compile time.
- As ‘java.lang.String’ and ‘java.lang.Integer’ is a more specific type than ‘java.lang.Object’, but between ‘java.lang.String’ and ‘java.lang.Integer’ none is more specific.
- In this case the Java is unable to decide which method to call.

```
public class Main
{
   public static void main(String args[])
   {
      String s1 = "abc";
      String s2 = s1;
      s1 += "d";
      System.out.println(s1 + " " + s2 + " " + (s1 == s2));
   }
}
```

- Output: abcd abc false
- Explanation : In Java, String is immutable, string s2 and s1 both pointing to the same string abc. And, after making the changes the string s1 points to abcd and s2 points to abc, hence false.

```
public class Main
{
   public static void main(String args[])
   {
      StringBuffer sb1 = new StringBuffer("abc");
      StringBuffer sb2 = sb1;
      sb1.append("d");
      System.out.println(sb1 + " " + sb2 + " " + (sb1 == sb2));
   }
}
```

- Output: abcd abcd true
- String buffer is mutable, both sb1 and sb2 both point to the same object.

```
class First
{
public First() {  System.out.println("a"); }
}
class Second extends First
{
   public Second()  {  System.out.println("b"); }
}
class Third extends Second
{
   public Third()   {  System.out.println("c"); }
}
public class Main
{
   public static void main(String[] args)
   {       Third c = new Third();  } }
```

- Output: a

  b

  c

- Explanation: While creating a new object of ‘Third’ type, before calling the default constructor of Third class, the default constructor of super class is called i.e, Second class and then again before the default constructor of super class, default constructor of First class is called.

```
class First
{
   int i = 10;
   public First(int j)
   {
      System.out.println(i);
      this.i = j * 10;
   }
}

public class Main
{
   public static void main(String[] args)
   {
      Second n = new Second(20);
      System.out.println(n.i);
   } }
```

```
class Second extends
First
{
   public Second(int j)
   {
      super(j);

System.out.println(i);
      this.i = j * 20;
   }
}
```

- Output: 10

  200

  400

- Explanation: Since in ‘Second’ class it doesn’t have its own ‘i’, the variable is inherited from the super class.
- Also, the constructor of parent is called when we create an object of Second.

```
class ThreadEx extends Thread
{
   public void run()
   {
      System.out.print("Hello...");
   }
   public static void main(String args[])
   {
      ThreadEx T1 = new ThreadEx();
      T1.start();
      T1.stop();
      T1.start();
   }
}
```

- Output: Run Time Exception
- Explanation: Exception in thread “main” java.lang.IllegalThreadStateException at java.lang.Thread.start
- Thread cannot be started twice.

```
class Main
{
   public static void main(String args[])
   {
      String s1 = new String("tulasi");
      String s2 = new String("tulasi");
      if (s1 == s2)
         System.out.println("Equal");
      else
         System.out.println("Not equal");
   }
}
```

- Output: Not equal
- Explanation: Since, s1 and s2 are two different objects the references are not the same, and the == operator compares object reference.
- So it prints “Not equal”, to compare the actual characters in the string .equals() method must be used.
- If you use s1.equals(s2) as condition then
- Output: Equal

```
class superClass
{
   final public int calc(int a, int b)   {   return 0; }
}
class subClass extends superClass
{
   public int calc(int a, int b)    {   return 1; }
}
public class Gfg
{
   public static void main(String args[])
   {
      subClass get = new subClass();
      System.out.println("x = " + get.calc(0, 1));
   }
}
```

- Output: Compiler Error
- Explanation: The method calc() in class superClass is final and so cannot be overridden.

```
class Main
{
  public static void main(String[] args)
   {
      Integer a = 128, b = 128;
      System.out.println(a == b);
      Integer c = 100, d = 100;
      System.out.println(c == d);
   }
}
```

- Output: false

  true

- Explanation: In the source code of Integer object we will find a method ‘valueOf’ in which we can see that the range of the Integer object lies from IntegerCache.low(-128) to IntegerCache.high(127).
- Therefore the numbers above 127 will not give the expected output.

# Thank You!