(weak)

Polymorphism:

Polymorphism (Method Overloading & Method Overriding) concepts:

~~Will be~~

Polymorphism → many forms

- water → solid, liquid, gas
- shapes → circle, triangle, square, etc
- sound → barking, roar, mew etc

Types:

1) Compile Time Polymorphism
   - Static Polymorphism
   - → method Overloading (achieve)

2) Run time Polymorphism
   - Dynamic Polymorphism
   - → method overriding

Method Overloading	Method Overriding
1) same name	1) same name
2) same class	2) different class
3) Different arguements	3) same arguements
- No of arguements	- No. of arguement
- Sequence of arguement	- Sequence of arguement
- Type of arguement	- Type of arguement.
	4) Inheritance IS-A relationship.

```
Class Test
{ void show()
  { s.o.p("1");
  }
  void show()
  { s.o.p("2");
  }
public static void main
(String[] args)
{ Test t = new Test();
  t.show()
}}
```

Q1) Can we achieve method overloading by changing the return type of method only?

Ans) In java, method overloading is not possible by changing the return type of the method only because of ambiguity.

Q2) Can we overload java main method?

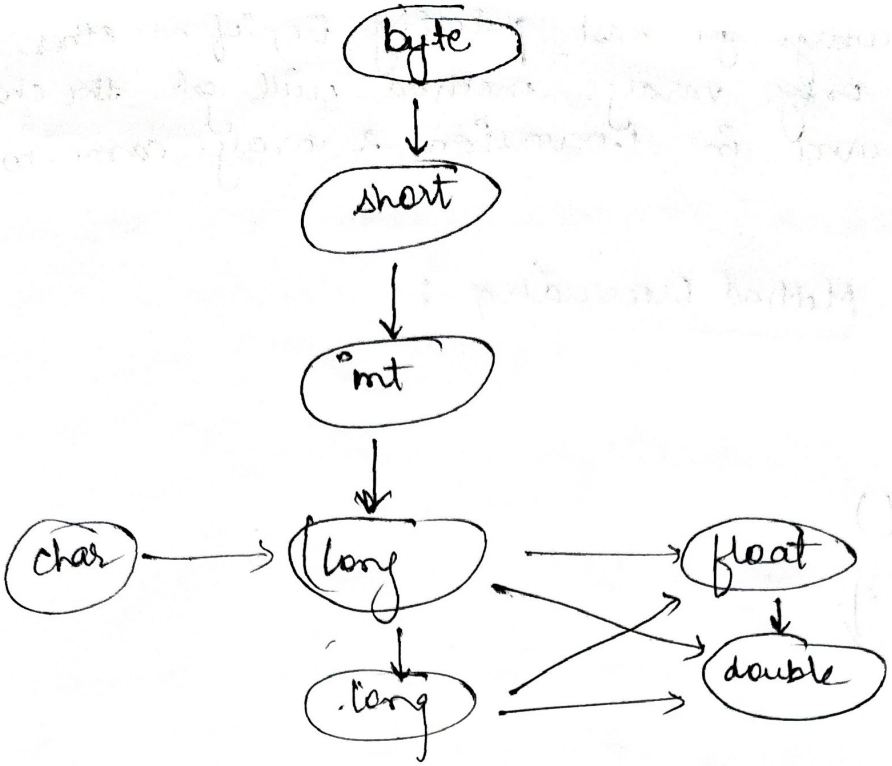
Ans) Yes, we can have any number of main methods in class by method overloading.
This is because JVM always calls main() method which recieves string array as arguments only.

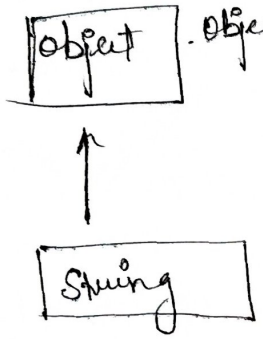
Example:

```
Class Test
{
public static void main (String args[])
 {
   System.out.println ("1");
   Test t = new Test();
   t.main(20);
 }
public static void main (int a)
 {
   System.out.println ("2");
   }
}
```

(3) Automatic Promotion:

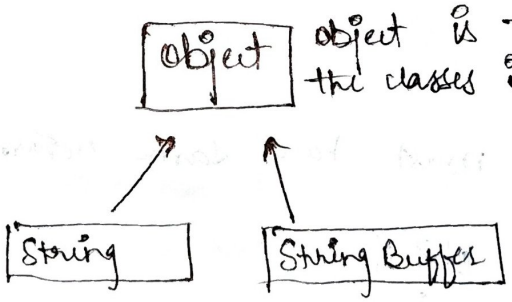
One type is promoted implicitly to no matching database is found. Below is the diagram:





Object is the parent class of all the classes in java.

While resolving Overloaded methods, Compiler always gives precedence for the child type argument than compared to with parent type argument.



object is the parent class of all the classes in java.

Var-args

The var-args allows the method to accept zero or multiple arguments. Before var-args either we use overloaded method or take an array as the method parameter but it was not considered good because it leads to the maintainance problem. If we don't know how many arguments we will have to pass in the method, var-args is the better approach.

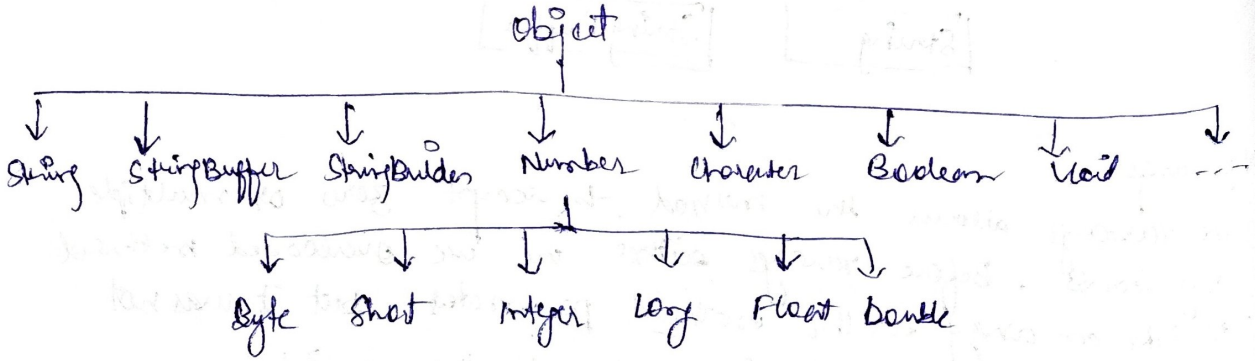
In general, varargs get least priority, i.e, if no other method matched, then only varargs method will get the chance, because int came in 1.0 version & varargs came in 1.5 version.

~~Types of Overriding~~ Method Overriding :

```
Class Test
 {
   void show ()
     {
       s.o.p("1");
     }
 }
 class xyz
   {
     void show()
       {
         s.o.p("2");
       }
     psvm ( )
       {
         ~~Test~~ Test t = new Test();
           t.show();
       }
```

Case 1 :

Q) Do overriding method must have same return type for subtype?



From Java 5.0 onwards, it is possible to have different return type for a overriding method in child class but child's return type should be sub-type of parent's return. This phenomena is known as covariant return type.

Case 2: (Overriding and Access Modifier)

The access modifier for an overriding method can allow more, but not less, access than the overridden method. For example, a protected instance made public, but not private, in the subclass. Doing so, will generate compile-time error.

Case 3: (Overriding & Exception Handling):

Case 4: (Overriding & Abstract Method):

Abstract methods in an interface or abstract class are meant to be overridden in derived concrete classes otherwise compile time error will be thrown.

Case 5: (Invoking overridden method from sub-class)

We can call parent class method in overriding method using super keyword.

Case 6:

Which method cannot override?

Final methods can not be overridden:

If we don't want a method to be overridden, we declare it as final.

Static methods can not be overridden:

(Method Overriding Vs Method Hiding)

When you define a static method with same signature as a static method in base class, it is known as method hiding.

Private methods can not be overridden:

Private methods cannot be overridden as they are bonded during compile time. Therefore we can't even override private methods in a subclass.

Case 7: Overriding and synchronized / strictfp method

The presence of synchronized / strictfp modifier with method have no effect on the rules of overriding, i.e, it's possible that a synchronized / strictfp method can override a non-synchronized / strictfp one and vice-versa.