* Polymorphism.

Polymorphism
Many forms

→ Ex :- water :- solid, liq, gas
(Ice)

* There are two types of polymorphism.

1. Compile time Polymorphism. (overloading)
   → Static Polymorphism
   → method overloading.
   → handle by compiler.

2. Run time Polymorphism (overriding)
   → Dynamic Polymorphism
   → method overriding
   → handle by JVM.

* Difference between method overloading & method overriding.

method overloading	method overriding.
→ multiple methods with same name	→ multiple methods with same name.
→ same class	→ different class.
→ different arguments.	→ same arguments.
- No. of arg.	- No. of arg.
- Seq. of arg.	- Seq. of arg.
- type of arg.	- type of arg.

* Important Interview questions for Polymorphism.

1. Can we achieve method overloading by changing the return type of method only ?

→
```
class test (int a)
{   void show (int a){
        S.o.v.t ("1");
    }
    String show (int a) {
        S.o.v.t ("2");
    }
    P.S.V.m (String [] args) {
        t. show (10);
    }
}
```

→ No, In java method overloading is not possible by changing the return type of the method only because of ambiguity.

2. Can we overload java main () method ?

→
```
class test {

    P.S.V.m (String [] args) {
        test t = new test ();
        S.O.V.t ("1");
    }
}
```

```
P.S.V.m (Int a) {
    S.o.V.t ("2");
}
}
```

-) to print "2" we have to call second main method. from first main method. like

```
test t = new test ();
t.main (20);
```

-) Here ~~meth~~ main method one will Automatically called

-) yes, we can have any number of main methods in a class By method overloading

-) this is because ~~of~~ jvm always calls main() method which receives String arrays as arguments only.

method overloading case-1

```
class test {

    Void show (Int a) {
        System.out.println ("Int method");
        System
    }

    Void show (String a) {
        System.out.println ("String method");
    }
    P.S.V.m (String [] args) {
        test t = new test ();
        t.show ('a');
    }
```

Page No.
Date : / /

y

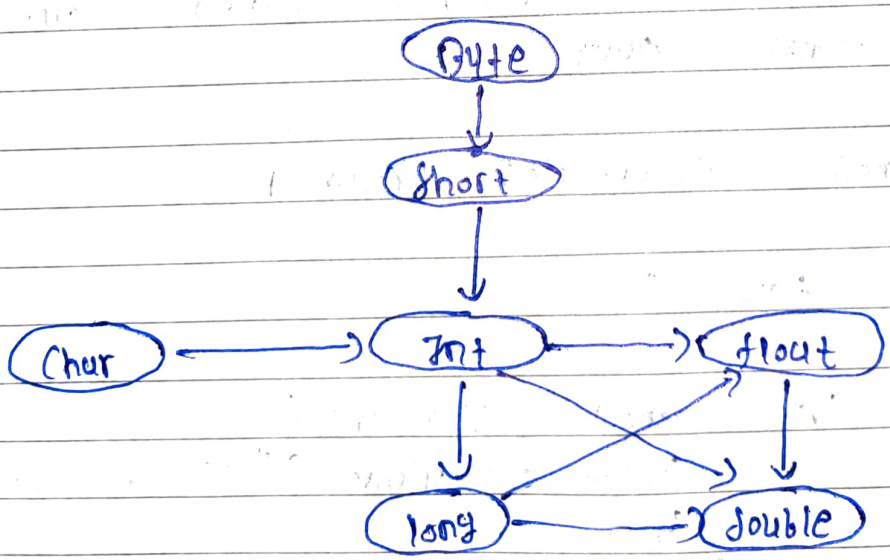
→ Here we pass Character But there is not single method who takes Character as argument.

→. But twist is that Here Int method runs. Because Here Automatic Promotion Activated.

Q- What is the Automatic Promotion.

→ Here one type is Promoted to another Implicity if no matching data-type is found.

→ diagram of Automatic Promotion.



Case-2-

→. " While resolving overloaded methods compiler will always give Precedence for the Child type argument than Compared with Parent type argument. "

Cal-3-

When two methods are overloaded in same class then and data types of arguments of both methods are same then

Case-3

```
Class test {

    Void show (int a)
{
        System.out.println ("int method");
}
    Void show (int ... a)
    {
        System.out.println ("from var args method");
    }

Public static Void main (String [] args)
{
        test t = new test ();
        t.show (10,20,30); // varargs method
        t.show (); // varargs method.
}
```

-). the varargs allows the method to Accept zero or multiple arguments Before varargs Either we use overloades method.

-). In general varargs get least priority. if no other method mathed. then only varargs will get the chance. because Int came in 1.0 version & varargs came in 1.5 version.

* Method Overriding.

-> Here Name of methods must be same
-> Both methods are in different class.
-> Same argument
  - No. of argument
  - type of argument
  - Seq of argument
-> Inheritance. (IS-A- Relationship)

-> Use of methodoverriding
(to change Implementations)

" method overriding allows a Subclass or Child class to provide a specific Implementation of a method that is already provided by one of its Superclasses or Parent classes. "

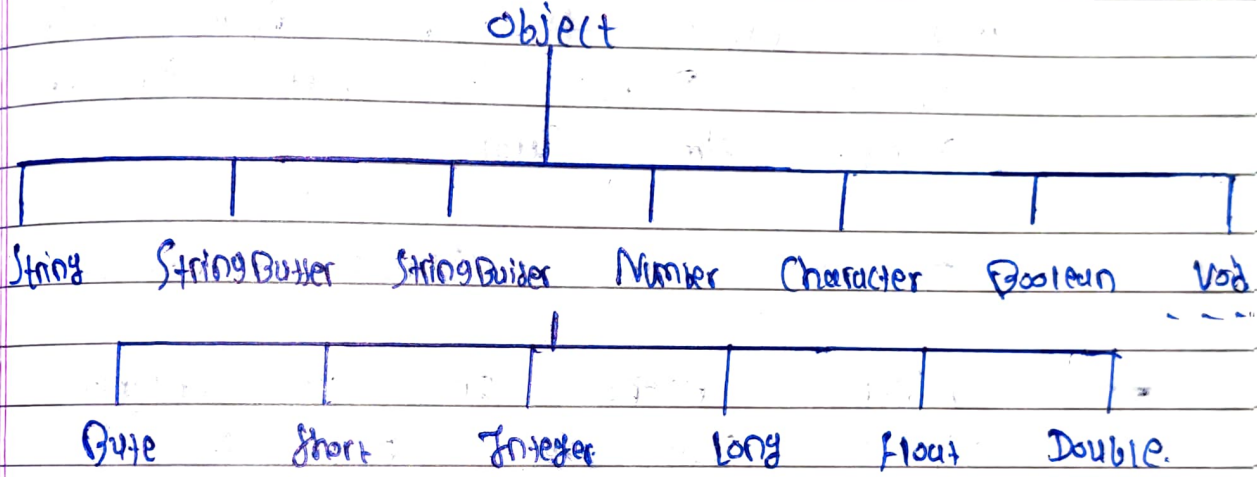
-> The Implementation in the Subclass overrides the Implementation in the Subclass. Superclass by providing a method that has same name, Same Parameters or signature, and Same return type as the method in the Parent class.

* Interview questions.

Q-1- Do overriding method must have same return type ?

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

-> this Phenomena is known as Covariant return type.




```
Class test {

    Void show ( ) {
        System.out.println ("1");
    }
Class XYZ Extends test {

    String show ( )
    {
        System.out.println ("2")
    }
```

-> Here I can use in test class show method (Void, Object ----) but not (Byte, Short, Integer, ----).

Q-2- overriding and access modifier.

-> The Access modifier for an overriding method can allow more, but not less, access than the overriden method.

-> For Example a Protected Instance method in the super-class can be made Public, but not Private. In the Subclass, Doing so, will generate Compile time Error.

Q-3 overriding and 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.

Q-5- Invoking Overridden method from sub-class.

-> We can call Parent class method in overriding method using super keyword.

```
super.methodName();
```

Q-6- Which methods Cannot override?

-> final methods can not be overridden
- if we don't want a method to be overridden, we declare it as final.

-> Static method can not be overridden :
- When You defines 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.
- As they are Bonded during compile time : therefore we can't even override private methods in a subclass.

Q.7- overriding and synchronized / strictfp method.

-> the presence of synchronized / strictfp modifier with method have no effects on the rules of overriding.

-> It's possible that a synchronized / strictfp method can override a non synchronized / strictfp one and vice-versa.