

Super Keyword

1.1) Super Key

polymorphism in java

(Method over)

Polymorphism

Compile time
(static)

Runtime

(Dynamic)

- Method overloading
- Handle by compiler
- Same name
- Same class
- Different Arguments
 - No. of Arguments
 - Seq. of Arguments
 - Types of Arguments

- Method overriding
- Handle by JVM
- Same name
- Diff. class
- Same Arguments
 - No. of Arguments
 - Seq. of "
 - Types of "
- Inheritance

Method overloading

(IS-A) relationship

class

test

void show ()

{ s.o.p ("1");

}

void show (String str)

{ s.o.p ("2");

}

public static void main (String args[])

{ test t = new test();

t.show ();

t.show ("3");

}



Q/ Can we overload java main() meth

```

class test
{
    void show (int a)
    {
        s.o.p ("1");
    }
    void show (String a)
    {
        s.o.p ("2");
    }
}

```

```

p.s.v.m (String args[])
{
    test t = new test();
    t.show (10);
}

```

```

p.s.v.m (String args[])
{
    s.o.p ("3");
}

```

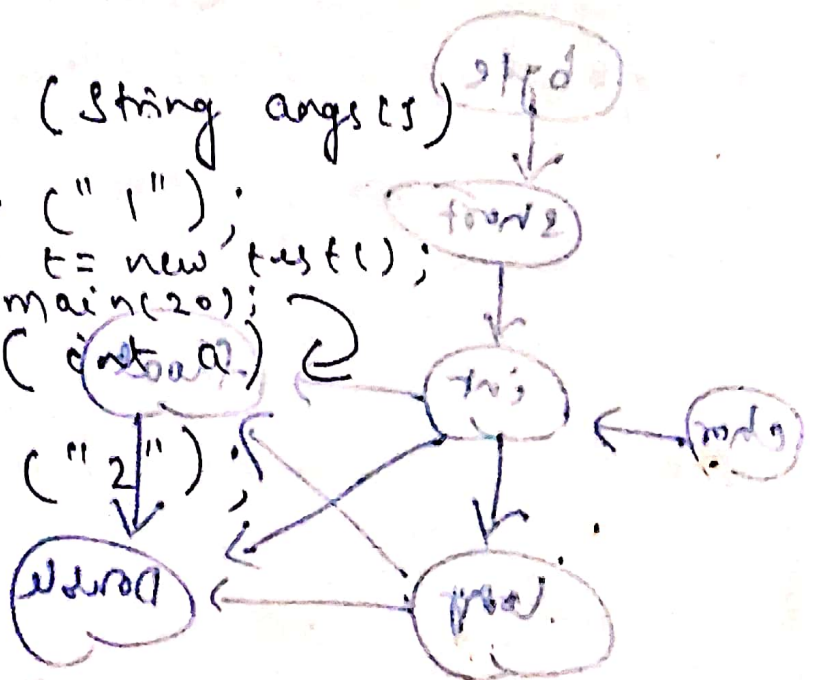
}

class test

```

p.s.v.m (String args[])
{
    s.o.p ("1");
}
test t = new test();
t.main (20);
p.s.v.m (int a)
{
    s.o.p ("2");
}

```



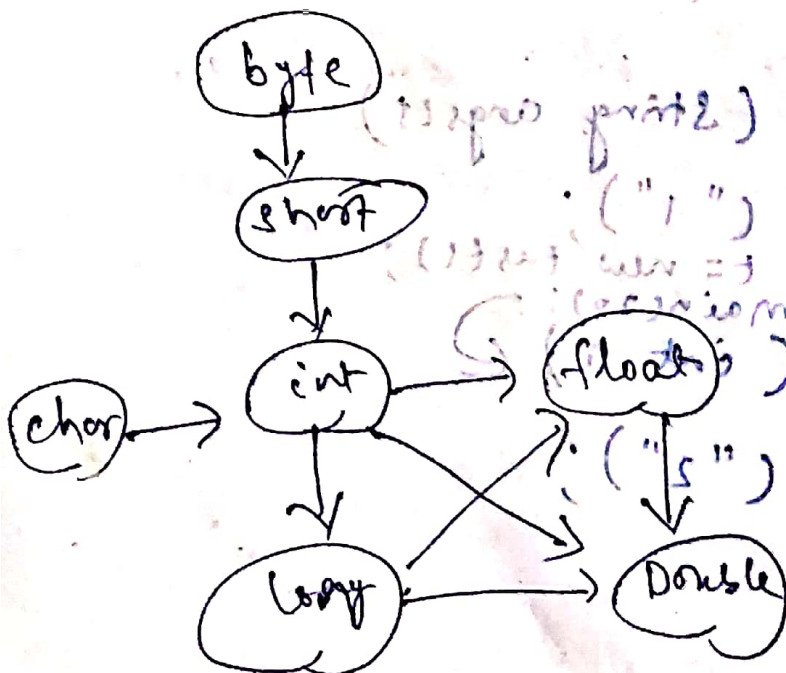
Q) Automatic Promotion

One type is promoted to another type implicitly if no matching data type is found.

e.g. class test

```
{  
    void show(int a) {  
        s.o.p ("int method");  
    }  
    void show(String a) {  
        s.o.p ("string method");  
    }  
}  
P.S.V.M (String args[]) {  
    test t = new test();  
    t.show(20);  
    t.show('a'); // character
```

↓
no match found.
Automatically converted
to integer.



Method overriding.

```
1 class test {
2     void show() {
3         s.o.p("1");
4     }
5 }

6 class xyz extends test {
7     void show() {
8         s.o.p("2");
9     }
10    p.s.v.m (String args[]) {
11        test t = new test();
12        t.show(); → 1.
13        xyz x = new xyz();
14        x.show(); → 2.
15    }
16 }
```

Diagram illustrating method overriding. A curved arrow points from the `show()` method in the `xyz` class to the `show()` method in the `test` class, indicating that the `xyz` class overrides the `show()` method of the `test` class.

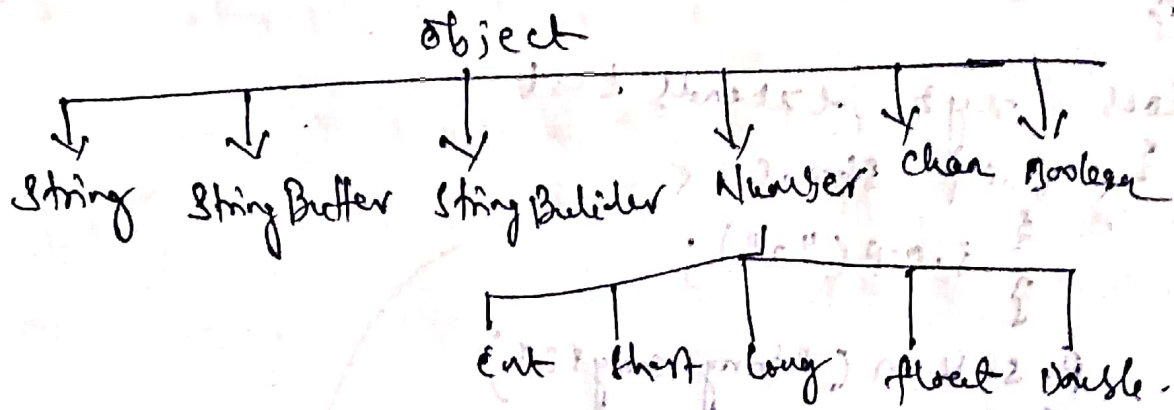
Do overriding method must have same return type? Co-variant return type.

```
class test {
    void show() {
        s.o.p("1");
    }
}

class xyz extends test {
    void show() {
        s.o.p("2");
    }
}
```

Diagram illustrating method overriding with a co-variant return type. A curved arrow points from the `show()` method in the `xyz` class to the `show()` method in the `test` class. The `show()` method in the `xyz` class is annotated with `String`, indicating that it returns a `String` value, which is a co-variant return type relative to the `void` return type of the `show()` method in the `test` class.

*) If Parent class Return type is parent datatype then we can override it in child class with child data type.



```

class test {
    {

```

```

    Object show() {
        {
            s.o.p("1");
            return null;
        }
    }

```

Do not override method with same signature

```

class xyz extends test {
    {

```

```

        String show() {
            {
                s.o.p("2");
                return null;
            }
        }
    }

```

```

    p.s.v.m() {
        {
            test t = new test();
            t.show();
            xyz x = new xyz();
            x.show();
        }
    }
}

```


Q2) Access Modifiers in Overriding

*) parent class Access modifier should be same or less than child class access modifier.

e.g.

class test

{

public

object show()

{

s.o.p("1");

}

}

class xyz extends test

{

public

string show()

{

s.o.p("2");

}

}

Protected
Private
Default

Protected

void show()

}

Public

void show()

}

greater than protected

Q3)

(100-0) marks awarded

(100) marks awarded