More On inheritance

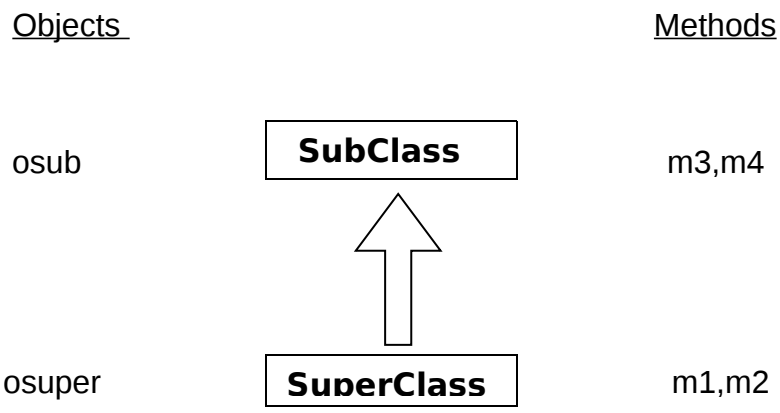
What you can do in subclass regarding methods:

- The inherited methods can be used directly as they are. You can write a new static method in the subclass that has the same signature as the one in the super class, thus hiding it. You can declare new methods in the subclass that are not in the super class.

```
class  SubClass extends SuperClass
{
        ……..
        ……...
        ……
}
```

All the methods (except private) of Superclass visible to SubClass.

Consider the example given below:



In this example super / base class has methods m1, m2 and sub/derived class has methods m3 and m4. superclass has object name osuper and subclass has object osub.

Now see the following:

- osub.m3(<args>)
- osub.m4(<args>)

These are fine because osub is an object of subclass and m3 and m4 are also methods of subclass. We can access member data and member function of a class using its object.

Similarly:

- osuper.m2(<args>)
- osuper.m1(<args>)

  are correct

see the followings:

- osub.m1(<args>)
- osub.m2(<args>)
- osuper.m3(<args>)
- osuper.m4(<args>)

In first two statements, osub is an object of derived/subclass and m1 and m2 are methods of base/super class. As we know that all the methods(except those with the scope private) of base class are also visible to subclass and we can access them by object of subclass. Both the statements are correct.

In third and last statement osuper is an object of super class,while m3 and m4 are methods of sub class, and we cannot access the methods of sub class by object of base class. Both the statement will not work.

Type Casting:

- ((superclass)osub).m1
- ((superclass)osub).m2

These will work because the final reference is of super class and we are accessing methods of super class.

- ((superclass)osub).m3

This won't work because final reference is of super class and we are accessing method of subclass(m3 is a method of subclass) by using reference of super class we cannot access method of subclass.

Now consider the example 2 :

Person is a base class has a derived class Student because Student is also a Person. Student extends Person

```
Class Student extends Person
{
        ……..
        ……..
};
```

```
Person person1 = new Student();
Student student1 = (Student) person1; // Explicit type casting
```

Runtime Type Mismatch Exception:

Even with explicit casting, you could still end up having a runtime error

● Example:

Let's assume **Student** class is a **child class** of Person class

Let's assume **Teacher** class is also a child class **of Person** class

```
Person person1= new Student ();
Person person2 = new Teacher ();
Student student1 = (Student) person1; // Explicit type casting
// No compile error, but runtime type mismatch exception if
Student student2 = (Student) person2
```

instanceof Operator:

it's a binary operator, takes an object and a class as operand and returns true and false.

See the following following from first example:

```
(osub  instanceof  Subclass )    // it will return true.
(osub instanceof  Superclass)  // it will also return true
```

Use instanceof Operator to Prevent Runtime Type Mismatch Error:

You can check the type of the object instance using ***instanceof*** before the type casting

Example:

```
Person person1 = new Student();
Person person2 = new Teacher();
    // Do the casting only when the type is verified
```

```
if (person2 instanceof Student) {
    Student student2 = (Student) person2;
     }
```

Super cosmic class: every class extends a class called Object class.

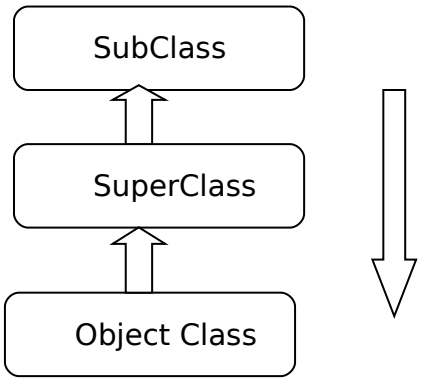
Object class called super cosmic class.

- Object Class is parent class of all classes – In Java language, all classes are sub classed (extended) from the Object super class
- Object class is the only class that does not have a parent class
- Defines and implements behavior common to all classes including the ones that you write.

Object class has following methods:

a) getClass() , b) equals() , c) toString() …….. etc.

Assume that you have a base class name SuperClass and derived class name SubClass.



You can override these methods like tostring() etc, that are also in Object class. Call to these methods will traverse the inheritance hierarchy from leaf to root, i.e. from the calling class up to Object class, until an appropriate match is found.