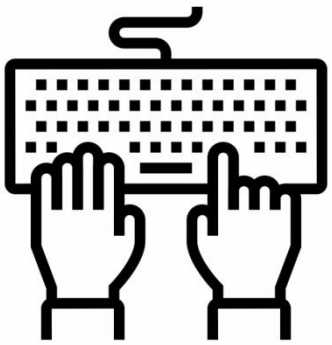
Specialized methods in java



Let us understand with the help of code…

```
class Plane //parent class
{
    void takeOff()
    {
        System.out.println("Plane is taking off");
    }
    void fly()
    {
        System.out.println("Plane is flying");
    }
    void land()
    {
        System.out.println("Plane is landing");
    }
}
class CargoPlane extends Plane //child class
{
    void fly()  // Overridden method
    {
        System.out.println("CargoPlane is flying at low heights");
    }
    void carryCargo() //specialized method
    {
        System.out.println("CargoPlane carries Cargo");
    }
}
class PassengerPlane extends Plane // child class
{
    void fly() // Overridden method
    {
        System.out.println("PassengerPlane is flying at medium heights");
    }
    void carryPassengers() //specialized method
    {
        System.out.println("PassengerPlane carries passengers");
    }
}
class FighterPlane extends Plane // child class
{
    void fly() // Overridden method
    {
        System.out.println("FighterPlane is flying at great heights");
    }
    void carryWeapons() //specialized method
    {
        System.out.println("FighterPlane is carrying weapons");
    }
}
```

```java
class Demo
{
    public static void main(String[] args)
    {
        CargoPlane cp = new CargoPlane();
        PassengerPlane pp = new PassengerPlane();
        FighterPlane fp = new FighterPlane();

        cp.takeOff();//Inherited method
        cp.fly();//Overridden method
        cp.land();//Inherited Method
        cp.carryCargo();//Specialized method

        pp.takeOff();
        pp.fly();
        pp.land();
        pp.carryPassengers();

        fp.takeOff();
        fp.fly();
        fp.land();
        fp.carryWeapons();
    }
}
```

Output:

```
Plane is taking off
CargoPlane is flying at low heights
Plane is landing
CargoPlane carries Cargo
Plane is taking off
PassengerPlane is flying at medium heights
Plane is landing
PassengerPlane carries passengers
Plane is taking off
FighterPlane is flying at great heights
Plane is landing
FighterPlane is carrying weapons
```

What did we see in the above example??

1. Inherited methods are such methods which are inherited from the parent class and used without any modification within the child class.
2. Overridden methods are such methods which are inherited from parent class and are modified as per the requirement of child class and used within the child class.
3. Specialized methods are such methods which are present only within the child class and not within the parent class.
4. Advantage of inherited methods is code reusability.
5. Advantage of overridden method is that the child class can modify the parent class method to satisfy its needs.
6. Advantage of specialized methods is that it increases the features of the child class when compared to parent class.

Amongst these four pillars of OOP, we have seen Encapsulation and Inheritance and virtual polymorphism. Now let's see how to achieve true polymorphism using inheritance.

1. **Abstraction** : Abstraction is the process of **showing only essential/necessary features** of an entity/object to the outside world and hide the other irrelevant information. (Will be discussed in detail in further classes.)
2. **Encapsulation** : Encapsulation means **wrapping up data** and member function (Method) together into a single unit i.e. class. Encapsulation automatically achieve the concept of **data hiding providing security to data** by making the **variable as private** and expose the property to access the private data which would be public.
3. **Inheritance** : The ability of **creating a new class from an existing class**. Inheritance is when **an object acquires the property of another object**. Inheritance allows a class (subclass) to acquire the properties and behaviour of another class (super-class). It helps to **reuse, customize and enhance the existing code**. So it helps to write a code accurately and reduce the development time.

4. **Polymorphism**: Polymorphism is derived from 2 Greek words: poly and morphs. The word **"poly" means many and "morphs" means forms**. So polymorphism means **"many forms".** A subclass can define its own unique behaviour and still share the same functionalities or behaviour of its parent/base class. A subclass can have their own behaviour and share some of its behaviour from its parent class not the other way around. A parent class cannot have the behaviour of its subclass.

In previous example if we change the main method to the following let's see what happens…

```
class Demo
{
    public static void main(String[] args)
    {
        CargoPlane cp = new CargoPlane();
        PassengerPlane pp = new PassengerPlane();
        FighterPlane fp = new FighterPlane();

        cp.fly();//one reference one behaviour
        pp.fly();//one reference one behaviour
        fp.fly();//one reference one behaviour

    }
}
```

We have three different references exhibiting 3 different behaviours. So now let's modify the main() and achieve polymorphism.

```
class Demo
{
    public static void main(String[] args)
    {
        CargoPlane cp = new CargoPlane();
        PassengerPlane pp = new PassengerPlane();
        FighterPlane fp = new FighterPlane();

        Plane ref; //Parent type reference
        ref=cp; //assigning child type reference to parent type
        ref.fly();//one reference one behaviour

        ref=pp; //assigning child type reference to parent type
        ref.fly();//same reference same behaviour

        ref=fp; //assigning child type reference to parent type
        ref.fly();//same reference same behaviour

    }
}
```

Creating parent type reference to child object is called loose coupling, through which we have achieved polymorphism. Whereas before we had tight coupling, where only child class's reference can access child class's behaviour. For example CargoPlane reference cp can access only CargoPlane behaviours and not of FighterPlane or PassengerPlane.

There is one limitation of parent type reference to child type. That is, using parent type reference we cannot directly access the specialized methods of child class.

With the same example as above here we are trying to access specialized methods through parent type reference **ref.** Let's see if we get any error, if yes then where is it pointing to…

```
50  class Demo
51  {
52      public static void main(String[] args)
53      {
54          CargoPlane cp = new CargoPlane();
55          PassengerPlane pp = new PassengerPlane();
56          FighterPlane fp = new FighterPlane();
57
58          Plane ref; //Parent type reference
59          ref=cp; //assigning child type reference to parent type
60          ref.fly();//one reference one behaviour
61          ref.carryCargo(); //accessing specialized method
62
63          ref=pp; //assigning child type reference to parent type
64          ref.fly();//same reference same behaviour
65          ref.carryPassengers(); //accessing specialized method
66
67          ref=fp; //assigning child type reference to parent type
68          ref.fly();//same reference same behaviour
69          ref.carryWeapons(); //accessing specialized method
70      }
71  }
```

Output

```
Demo.java:61: error: cannot find symbol
                ref.carryCargo(); //accessing specialized method
                   ^
  symbol:   method carryCargo()
  location: variable ref of type Plane
Demo.java:65: error: cannot find symbol
                ref.carryPassengers(); //accessing specialized method
                   ^
  symbol:   method carryPassengers()
  location: variable ref of type Plane
Demo.java:69: error: cannot find symbol
                ref.carryWeapons(); //accessing specialized method
                   ^
  symbol:   method carryWeapons()
  location: variable ref of type Plane
3 errors
```

We can see that we have got compile time error wherever we tried accessing specialized method using parent type reference.

So now let's see how to overcome this…


```
class Demo
{
    public static void main(String[] args)
    {
        CargoPlane cp = new CargoPlane();
        PassengerPlane pp = new PassengerPlane();
        FighterPlane fp = new FighterPlane();

        Plane ref; //Parent type reference
        ref=cp; //assigning child type reference to parent type
        ref.fly();//one reference one behaviour
        ((CargoPlane)(ref)).carryCargo(); //Downcasting

        ref=pp; //assigning child type reference to parent type
        ref.fly();//same reference same behaviour
        ((PassengerPlane)(ref)).carryPassengers(); //Downcasting

        ref=fp; //assigning child type reference to parent type
        ref.fly();//same reference same behaviour
        ((FighterPlane)(ref)).carryWeapons(); //Downcasting
    }
}
```

Output:

```
CargoPlane is flying at low heights
CargoPlane carries Cargo
PassengerPlane is flying at medium heights
PassengerPlane carries passengers
FighterPlane is flying at great heights
FighterPlane is carrying weapons
```

We have overcome it by changing the type of ref to respective child type reference explicitly. And this is called as Downcasting.