MAJOR OOP CONCEPTS

December 22, 2015

.

- Inheritance - Abstraction - IS-A / HAS-A
- PolyMorphism - Dynamic Binding
- Overriding
- Overloading
- Abstract Classes
- Interfaces

1 Inheritance

→ *The concept of Inheritance is fundamental to Object Orientation.*

→ *The concept is based on the fact that, we will abstract out all the features of subclasses and place them inside the superclass.*

eg 1: A car has the following properties. .

- mileage
- engine - capacity
- occupancy
- purpose

So, a car can be modelled in java like

```
Class Car
 {
    int occupancy;
    int engine-capacity;
    float mileage;
    void purpose()
       {
```

```
            System.Out.Println ("Family trip or Cab Service");
        }
    void HatchbackOrSedan ()
        {
            System.Out.Println ("HatchbackOrSedan ");
        }
}
```

Above program follows exclusive to Cars

Similarly a bike can be modelled as follows

```
Class Bike
 {
    int occupancy;
    int engine-capacity;
    float mileage;
    void purpose()
        {
            System.Out.Println ("Go out with friends or GF");
        }
    void GearOrWithoutGear ()
        {
            System.Out.Println (" Geared / Without Geared");
        }
}
```

Above program follows exclusive to bike

$\rightarrow$ *So in the above illustration, we can identify that..*

- mileage
- engine - capacity
- occupancy
- purpose()

Above properties common for both CAR and BIKE

HatchbackOrSedan () $\rightarrow EXCLUSIVE\ TO\ CAR$
GearOrWithoutGear () $\rightarrow EXCLUSIVE\ TO\ BIKE$

$\rightarrow$ *When you find such scanacious, we will abstract out all the common features and place them inside a superclass.*

$\rightarrow$ *This process is called "abstraction".So the above illustration can be modelled as follows.*

```
Class Vehicle
 {
    int occupancy;
    int engine-capacity;
    float mileage;
```

Above lines are for Abstraction

```
    void purpose()
      {
        System.Out.Println ("Some purpose");
      }
 }
Class Car extends Vehicle
      {
    void purpose()
      {
        System.Out.Println ("Family trip or Cab Service");

        }
    void HatchbackOrSedan ()
      {
        System.Out.Println ("HatchbackOrSedan ");
      }
 }
Class Bike extends Vehicle
      {
    void purpose()
      {
        System.Out.Println ("Go out with friends or GF");
```

```
    }
  void GearOrWithoutGear ()
    {
      System.Out.Println (" Geared / Without Geared");
    }
}
```

2 Observations

→ *Variables and methods which are common to all subclasses are taken out and placed in the superclass,"vehicle".*

→ *Method purpose() is performing different things in Car and Bike subclasses.So you abstracted out and kept in the superclass.You gave some "dummy" body to it as following :*

```
void purpose()
{
  System.Out.Println ("Some purpose");
}
```

Above program is dummy body given in superclass vehicle

→ *You later overrided in subclasses.*
→ *The class specific properties such as*

HatchbackOrSedan () → *EXCLUSIVE TO CAR*

GearOrWithoutGear () → *EXCLUSIVE TO BIKE*

ARE NOT PLACED IN THE SUPERCLASS.

→ *Now we will extend it to class Animal and lets see, how inheritance works for our abstraction.*

Now let us observe the Dog,Cat,Horse and abstract the features to a animal.

```
Class Dog
 {
   int weight;
   int colour;
   int age;
   void talk()
```

```
        {
            System.Out.Println ("bow");
        }
    void eat()
        {
            System.Out.Println ("bones");
        }
    void provide Security()
        {
            System.Out.Println ("Security to houses");
        }
}
```

Above program is exclusive to Dog Class

```
Class Cat
 {
    int weight;
    int colour;
    int age;
    void talk()
        {
            System.Out.Println ("Meow");
        }
    void eat()
        {
            System.Out.Println ("drinking Milk");
        }
    void StealMilk()
        {
            System.Out.Println ("Steal the Milk");
        }
}
```

Above program is exclusive to Cat Class

```
Class Horse
 {
    int weight;
    int colour;
    int age;
    void talk()
       {
          System.Out.Println ("Grass ");
       }
    void eat()
       {
          System.Out.Println ("Eating Grass");
       }
    void transport()
       {
          System.Out.Println ("Pulling Carts");
       }
 }
```

→ *Let us abstract the common features out and keep it inside Animal.*

```
Class Horse
 {
    int weight;
    int colour;
    int age;
    void eat()
       {
          System.Out.Println ("Eat Something");
       }
    void talk()
       {
          System.Out.Println ("Talking");
       }
 }
```

All these things are common to Dog,Cat,Horse.So abstracted them to Animal

```
Class Cat extends Animal
 {
    void talk()
       {
          System.Out.Println ("Meow");
       }
    void eat()
       {
          System.Out.Println ("drinking Milk");
       }
    void StealMilk()
       {
          System.Out.Println ("Steal the Milk");
       }
 }
Class Dog extends Animal
    {
    void talk()
       {
          System.Out.Println ("bow");
       }
    void eat()
       {
          System.Out.Println ("bones");
       }
    void provide Security()
       {
          System.Out.Println ("Security to houses");
       }
 }
Class Horse extends Animal
    {
```

```
        void talk()
          {
            System.Out.Println ("Grass");
          }
        void eat()
          {
            System.Out.Println ("Eating Grass");
          }
        void transport()
          {
            System.Out.Println ("Pulling Carts");
          }
    }
```

→ *After understanding what is inheritance, let us see some examples how to use them.*

eg 1:

```
    Class A
    {
      int i;
      void b()
        {
          System.Out.Println ("Hi");
        }
  Class B extends A
    {
      int j;
      int i; → gives error
      void a()
        {
          System.Out.Println ("Hiiiii");
        }
  Class Test
    {
      public static void main (String[] args)
        {
```

```
        A    a = new    A();
             a.i = 10;
        B    b = new    B();
             b.j = 20;
             b.i = 20;          → inherited
             b.b() = 200;       → inherited
             a.a();             → willfail
             a.j() = 200;       → willfail
        }
    }
```

→ *Private variables can't be inherited*)

```
Class A
    {
      private int i;
      private int a();
        {
          return 10;
        }
    }
Class Test
    {
      public static void main (String[] args)
        {
         B    b = new    B();
              b.i = 200;        → not inherited
              b.a();            → not inherited
        }
    }
```

→ *final classes can't be subclassed.*

```
final class A {}
class B extends A {} → error
```

→ *One of most famous final library classes is string class.*

→ *Then what about classes like Exception, Tread*??

Check out whether they are final or not?

Think why String class is final and these two classes are not final?

3 IS-A, HAS-A RELATIONSHIPS

→ *With Inheritance, there "IS−A" RELATIONSHIP that comes into play.*

→ *When class A extends classB, then we say*

A "IS - A" B

eg 1: Class B { }

Class A extends B { }

A "IS - A" B

eg 2: Class Animal { }

Class Dog extends Animal { }

Class Cat extends Animal { }

Class GermanShephared extends Dog { }

Class Lebrador extends Dog { }

The following are TRUE	The following are FALSE
Dog IS-A ANIMAL	ANIMAL IS-A Dog
Lebrador IS-A ANIMAL	Lebrador IS-A Cat
Cat IS-A ANIMAL	GermanShephared IS-A Lebrador
Lebrador IS-A Dog	
Cat IS NOT A Dog	

eg 3: Apply for Exception Hierarchy.

USE OF "IS-A" RELATIONSHIP

Instead of superclass, you can always use anything which satisfies "IS-A" relation.

eg 4: Class A

```
{
  C a()
    {
      return new E();  → E   IS − A   C.
    }
```

```
}
Class C { }
  Class D extends C { }
  Class E extends D { }
```

eg 5:

```
Class A{ }
  Class B extends A{ }
  Class C
  {
    A    a()
    {
      return new A();
    }
  }
  Class D extends A
  {
    B    a()   → perfect overriding.Since B IS − A A
    {
      return new B();
    }
  }
```

HAS - A RELATIONSHIP

→ $Ithasnothingtodowith"Inheritance".$

→ $Observethefollowing$

```
Class A
  {
    int b;
    string s;
    B b;
  }
Class B
```

{

int k;

float f;

}

Observe that class A **HAS**

.

- int b
- string s
- B b

So Class A HAS-A B.
Class A HAS-A String.

→ *In such cases, we will say that there is a"tightCoupling"between A and B.*

→ *Tight Coupling means having strong relationship.In the above case, everytime object A is created, it also contains the reference variable b, which points to"B"object.*

→ *Similar word is"Cohesion".*

→ *Observe the levels of Coupling.*

Class A
{ }
Class B{ }
-
-
-
-
-
I
	Class A
	{
	B b;
	}
	Class B{ }
	-
	-
	-
	II
		Class A
		{
		B b;
		}
		Class B
		{
		A a;
		}
		III

Order of Coupling from "Loose" to "Tight".

I <II <III
Case III is very tight coupling.