PSLG Week 8

Ben & Amy

DLSH

ICTLC Online QR Code:

Github

Agenda

- Inheritance (What it do?)
- Polymorphism

P.S. Big words I know... BUT I promise they’re not as scary as they sound

Inheritance

And no I’m not talking about your grandmothers will.

Inheritance in an Object Orientated programming paradigm like java (Something with objects and classes that are made from those objects) is the ability for a newly constructed class to inherit features and qualities of its class its inheriting from.

The class that’s inheriting features is usually called the “*Child*” and the one that is giving those features is the *“parent”*

Note : A feature is anything contained within that class, Like a variable or method

Inheritance Syntax

Normal class instantiation Syntax

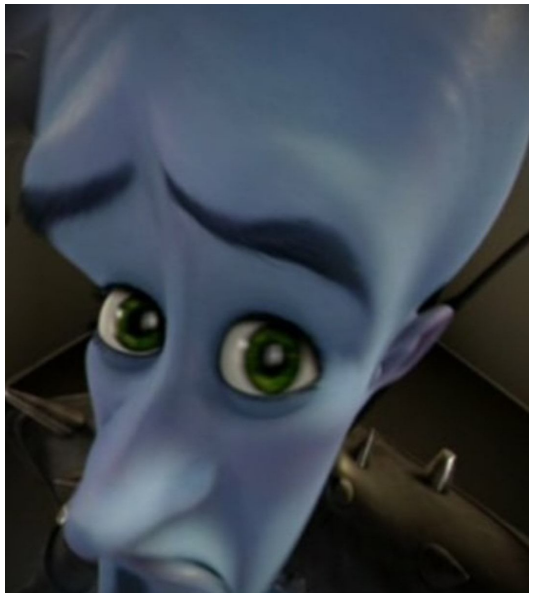
```java
public class Car   no usages
{
    // Global variables
    private String reg;  1 usage
    private String model;  1 usage
    private String color;  1 usage

    // Constructor
    public Car(String reg, String model, String color)  no usages
    {
        this.reg = reg;
        this.model = model;
        this.color = color;
    }

    // Some method
    public void horn()  no usages
    {
        System.out.println("Meep Meep Meep Meep");
    }
}
```

Inheritance Syntax

Silly Inheritance class :



```java
public class Porsche extends Car  no usages
{
    // 1. Different attributes
    private int price;  1 usage

    public Porsche(String reg,String color,String model, int price)  no usages
    {
        // 2. Can use its parents constructor in its own
        super(reg,color,model);
        this.price = price;
    }

    // Can override pre-existing methods in other classes
    @Override  no usages
    public void horn()
    {
        System.out.println("Meep Meep But fancily");

    }

}
```

Inheritance problem

Say you are opening up your own zoo that houses many different animals and you want a program that keeps track of some of them. All animals have a name and a sound.

Mammals have their own attribute – fur colour.

Birds also have their own attribute – they can fly.

The three tasks you must do to implement this program are:

1. Create a base class Animal that constructs the attributes of an animal
2. Create a subclass for Mammals that inherits from Animal
3. Create a subclass for Birds that inherits from Animal

Expected output: Dog makes a Woof sound. It has brown fur.

Parrot makes a Squawk sound. It can fly.

Inheritance solution

```
2 usages  2 inheritors
class Animal {
    5 usages
    protected String name;
    2 usages
    protected String sound;

    2 usages
    public Animal(String name, String sound){
        this.name = name;
        this.sound = sound;
    }

    2 usages  1 override
    public void makeSound(){
        System.out.println(name + " makes a " + sound + " sound.");
    }
}
```

```
2 usages
class Bird extends Animal{
    2 usages
    protected  boolean canFly;

    1 usage
    public Bird(String name, String sound, boolean canFly){
        super(name, sound);
        this.canFly = canFly;
    }

    1 usage
    public void flightAbility(){
        if(canFly){
            System.out.println(name + " can fly");
        }else{
            System.out.println(name + " cannot fly");
        }
    }
}
```

Inheritance solution contd.

```
2 usages
class Mammal extends Animal{
    2 usages
    protected String furColour;

    1 usage
    public Mammal(String name, String sound, String furColour){
        super(name, sound);
        this.furColour = furColour;
    }

    1 usage
    public void furColour(){
        System.out.println(name + " has " + furColour + " fur colour");
    }
public class Zoo{
    public static void main(String[] args) {
        Mammal dog = new Mammal( name: "Dog",  sound: "Woof",  furColour: "Brown");
        Bird crow = new Bird( name: "Crow",  sound: "Kaw",  canFly: true);

        dog.makeSound();
        dog.furColour();

        crow.makeSound();
        crow.flightAbility();
    }
}
```

Polymorphism

Polymorphism is the ability of a single interface or method to take multiple forms. It enables different classes to respond uniquely to the same method call. This can happen in two ways:

1. **Method Overriding** (Runtime Polymorphism) – A subclass provides a specific implementation of a method from its superclass, determined at runtime.

2. **Method Overloading** (Compile-Time Polymorphism) – Multiple methods in the same class share the same name but have different parameters.

Polymorphism syntax

Example of overidding

```
// Some method
public void horn() { System.out.println("Meep Meep Meep Meep"); }
```

```
// Can override pre-existing methods in other classes
@Override  no usages
public void horn()
{
    System.out.println("Meep Meep But fancily");

}
```

Polymorphism question

You are Loki! A renowned trickster and prankster within the norse pantheon and you really want to play a prank on your nuisance of a brother Thor by pretending to be him using your *poly-juice* potion (self explanatory).

Create a Thor class that has a weapon data field, a title data field (i.e. god of thunder) and a datafield to denote whether this is the real thor or not.
Write a void method that prints out the message “I am Thor!” + title in the class
Create a Loki class with similar data fields except you must set the real thor field to false in the constructor and Override the void method to now say “HAHA I got you!”.

Solution

Thor class

```
15
16  class Thor  4 usages  1 inheritor
17  {
18      public String weapon;  1 usage
19      public String title;  2 usages
20      public boolean isReal;  1 usage
21
22      public Thor(String weapon, String title, boolean isReal)  2 usages
23      {
24          this.weapon = weapon;
25          this.title = title;
26          this.isReal = isReal;
27      }
28
29      public void title()  2 usages
30      {
31          System.out.println("I am Thor!" + title);
32      }
33  }
```

Solution

Loki class

```
34  class Loki extends Thor  1 usage
35  {
36      public Loki(String title, String weapon)  1 usage
37      {
38          super(weapon,title, isReal: false);
39      }
40
41      @Override  2 usages
42      public void title()
43      {
44          System.out.println("HAHAHA GOT YOU");
45      }
46
47  }
48
```

Solution

Main driver

```
public class Problem2
{
    // Main driver
    public static void main(String[] args)
    {
        Thor thor = new Thor( weapon: "Mjolnir", title: "God of Thunder!", isReal: true);
        Thor loki = new Loki( title: "God of Mischief", weapon: "Lævateinn");

        thor.title();
        loki.title();
    }
}
```

Final Problem

You took a time machine and are now 4 years in the future!!! in your first software development job and absolutely terrified because you know nothing. Your boss has asked you to make a system simulates the transaction of Items in the store.

Luckily you just did this PSLG session so you know everything you need to meet his requirements !

His requirements are the following

Final problem

1. To have a class that represents every item in the shop.
   Each item has a :
   a. Name
   b. Price
   c. Quantity
   Each item can also be sold, this will decrease the quantity of the item.

2. To have 2 subclasses :

   1. That represents a Item from the deli (Like a delicious chicken roll) which doesn't have a quantity limit neither does its quantity decrease from purchase.

   2. An item that is "Sold Out" which has a quantity of 0 and prints an error message when someone attempts to buy it.

Solution : Item Class

```java
class Item  6 usages  2 inheritors
{
    String name;  3 usages
    double price;  3 usages
    int quantity = 0;  4 usages

    Item(String name, double price, int quantity)  3 usages
    {
        this.name = name;
        this.price = price;
        this.quantity = quantity;
    }

    public void purchase()  3 usages  2 overrides
    {
        if (quantity <= 0)
        {
            System.out.println("Sold out!");
        }
        else
        {
            System.out.println("You purchased " + name + " with price " + price + " and quantity " + quantity)
            quantity--;
        }
    }
}
```

Solution : Chicken Roll class

```
47  class ChickeyRoll extends Item  1 usage
48  {
49      ChickeyRoll(String name, double price, int quantity)  1 usage
50      {
51          super(name, price, quantity);
52      }
53
54      @Override  3 usages
55      public void purchase()
56      {
57          System.out.println("Chickey rolled! You purchased " + name + " with price " + price );
58      }
59  }
```

Solution: Sold Out Item Class

```
61  class SoldOutItem extends Item  1 usage
62  {
63      SoldOutItem(String name, double price)  1 usage
64      {
65          super(name, price,  quantity: 0);
66      }
67
68      @Override  3 usages
69      public void purchase(){
70          System.out.println("Sold out!");
71      }
72  }
```

Solution : Driver Class

```
3
4   public class Problem3
5   {
6       // Main driver
7       public static void main(String[] args)
8       {
9           Item soda = new Item( name: "Coke", price: 2.70, quantity: 30);
10          Item chickenRoll = new ChickeyRoll( name: "Chicken Roll", price: 5.95, quantity: 30);
11          Item soldOut = new SoldOutItem( name: "Monster", price: 3.20);
12
13          soda.purchase();
14          chickenRoll.purchase();
15          soldOut.purchase();
16
17      }
18  }
```