Object Oriented Systems Analysis & Design using Java & UML

Lecture 3 : Inheritance - Revision

Lecture Contents

- Inheritance
  - Super Class (parent class)
  - Subclass (child class)
  - Generalization
  - Specialization
  - Java’s Object Hierarchy

Inheritance

- By making use of **Inheritance** we can create **new classes** from already **existing ones** by **extending them** with new attributes and methods.

- Using inheritance we can express object relationships such as “an athlete is a person” or “a book is a document”.

Inheritance

- In Java inheritance is introduced by creating ***subclasses*** of already existing classes.
- Subclasses are created by using the ***extends*** keyword when defining a new subclass.
- By using inheritance we can create objects that are only partly equal to other objects. (this will become more apparent later on!)
- The best way to understand inheritance is to have a look at an example program.

Inheritance

```
public class Building
{
    // Data members
    private double length,width;
    private int nofloors, lastrenovation;

    // Default constructor
    public Building()
    {
        length = 0;
        width = 0;
        nofloors = 0;
        lastrenovation = 0;
    }
    // Usual set & get methods should go here

    // This method will return the total
    // area of the  building
    public double area()
    {
        return length * width * nofloors;
    }
}
```

- The class above models a general object called building.

Inheritance

- The class has data members which model the buildings physical attributes, length, width, and number of floors.
- It also has a data member which stores the year in which the building was last renovated.
- I have omitted the usual set and get methods as we will assume that they will be written at some stage.
- Lastly the class has a method called area which is a general thing that can be done with all types of buildings.

Inheritance

- Lets suppose for a moment that we wanted to model a specific type of building, lets say a house that people could live in.

- A house
  - has a length, width and a number of floors
  - may have been renovated
  - has an area which is given by length*width*No. floors
  - will usually be insulated
  - will usually have a number of bedrooms

- We already have a class that models some of the above attributes and behaviours

- So why not extend our building class to produce a house class.

Inheritance

- Lets have a look at the code to do this

Subclass Super Class

```
public class House extends Building
{
    private boolean insulated;
    private int nobedrooms;

    public void insulate()
    {
        insulated = true;
    }
}
```

Extra data members

Extra methods

Inheritance

- Note that when we **define a subclass** we use the keyword **"*extends*"** .

- The following template can be used when defining a subclass

```
class subclass_name extends superclass_name
{
        // declarations of additional instance variables
        // definitions of additional methods
}
```

Inheritance

- An object of type house will have 6 data members
  - length (inherited from Building)
  - width " "
  - nofloors " "
  - lastrenovation " "
  - insulated (own data member)
  - nobedrooms " "

Inheritance

- An object of type house will have the following methods

– setLength	getLength	(inherited from building)
– setWidth	getWidth	“ “
– setNoFloors	getNoFloors	“ “
– setLastRen	getLastRen	“ “
– area		“ “
– setInsulated	getInsulated	(own method)
– setNoBedRooms	getNoBedRooms	“ “
– Insulate		“ “

Inheritance

- A class that is a subclass can in turn be a superclass
  - (in fact any class in Java is a potential superclass)
- For example we can define a class that models houses with several flats.
- We can extend our house class and add a data member to store the number of flats in the building.
- The next slide shows an example of this class.

Inheritance

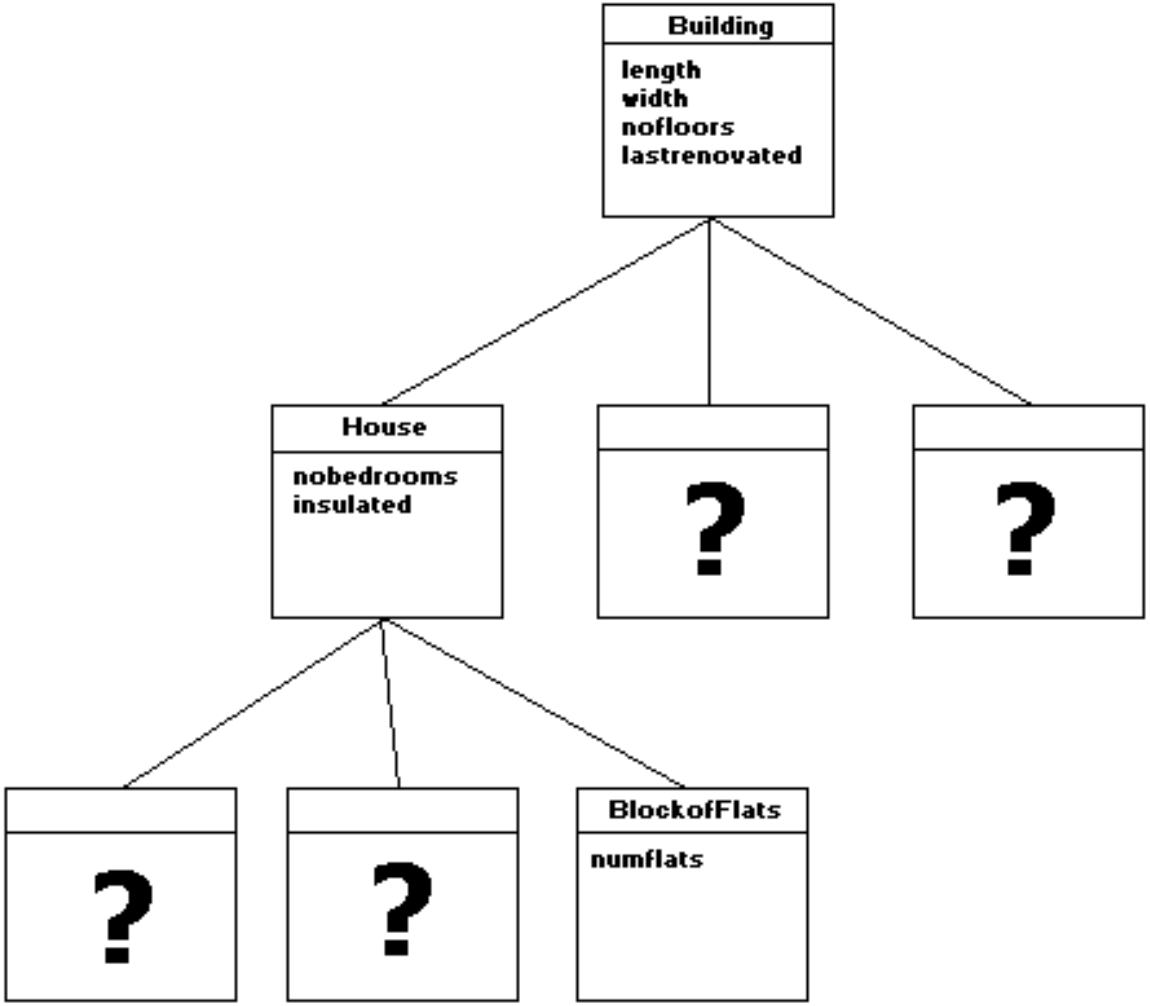
```
public class BlockofFlats extends House
{
    int numflats;

    static final double rentPerM2 = 100;

    public double calcRent()
    {
        return area() * rentPerM2;
    }
}
```

- As each flat will be rented, we have included and constant variable called rentPerM2 so that rent can be calculated.
- A calcRent method has also been added so that we can call it to return the amount of rent owed.

Inheritance

- We can think of **inheritance** as a **kind of family tree for classes** where each **child class inherits** the data members and methods **from its parent**.
- Any class that has no descendants is called a leaf class
- A class that has descendants will itself be a parent and can be seen as a branch of the family tree.

Inheritance



Inheritance

- In Java it is possible to **create a class from which there can be no subclasses create**. This is achieved by using the keyword ***final*** when declaring a class.

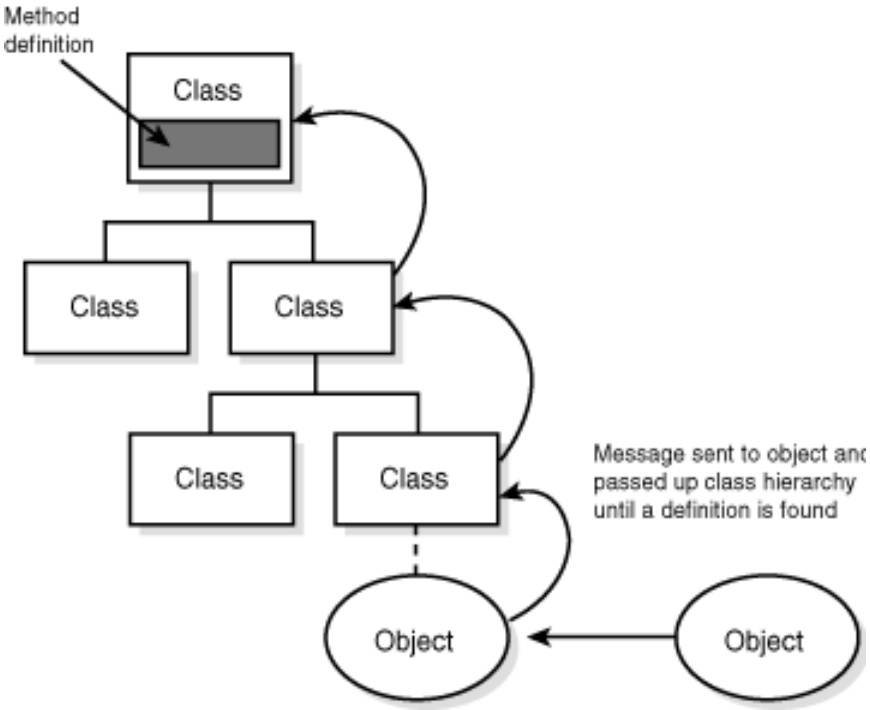
```
public final class C
{
    ..........
}
```

- This means that the class C cannot be extended.

```
public class child extends C
{
    ..........
}
```

Inheritance

- So what happens when you call an inherited method such as area ? How does Java know where to find the method ?
- Java starts at the bottom of the hierarchy and works its way upwards towards the root until it finds a method that matches the call.



Inheritance

- **Generalisation**
  - Generalisation is when we take two classes with similar attributes and generalise to give a superclass.
  - For example book and magazine might give literature
  - **Captures similarities between classes**
- **Specialisation**
  - This is when take an existing class and create subclasses by extending it.
  - For example when we create a House from the Building class we were specialising
  - **Captures differences between similar classes**