CS 162FZ: Introduction to Computer Science II

Lecture 12

Inheritance

Dr. Chun-Yang Zhang

Introduction

- With classes and objects we saw how we can very efficiently capture the attributes and functionality of a real-world object in code.
- This allows us to re-use that code again and again without having to re-write it: **WORM - Write Once Read Many** times.

Inheritance

- What if we wanted to write similar code and only change a small part of it?
- Should we re-write all the bits that don't change as well? No! .
- With inheritance, we can keep the functionality and attributes that we do need and then write new code for the extra stuff
- Java allows us to reuse class definitions and extend functionality by allowing one class (the child or subclass) to **inherit** from another class (the parent or superclass).

Inheritance

- This is very important as it allows developers to create software that is based on previously proven super classes.

Inheritance: Example

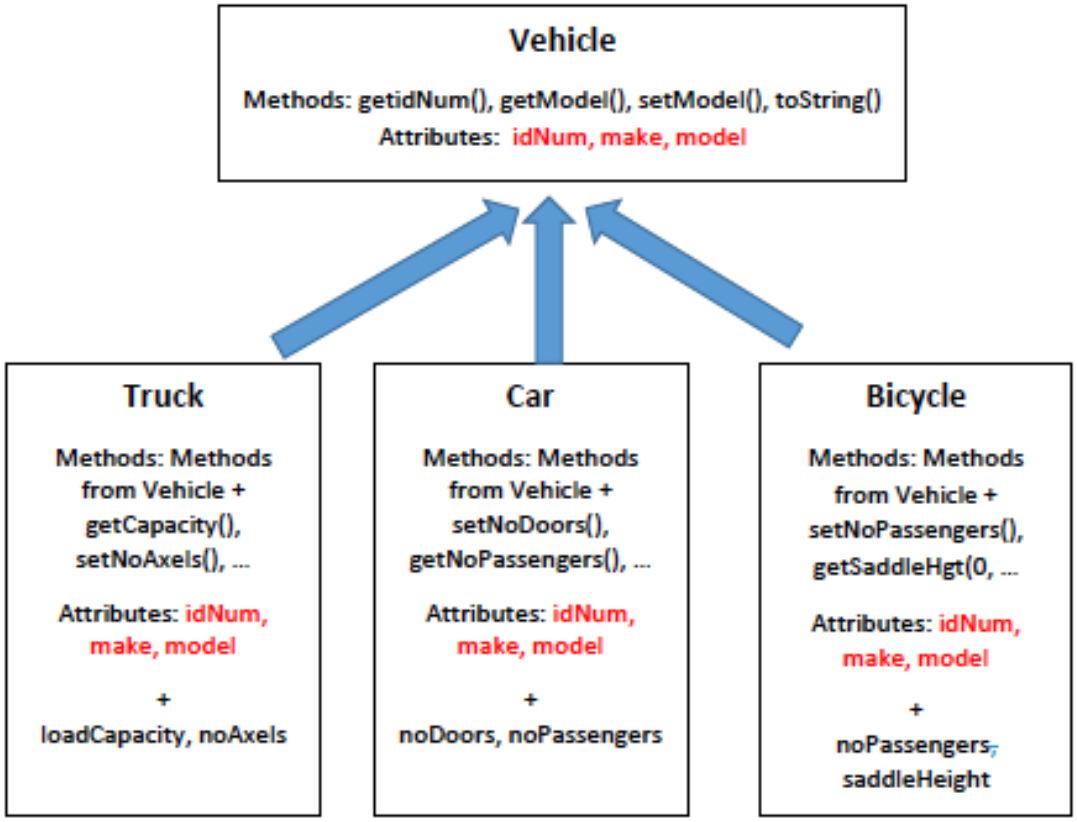
- Consider the case of representing a vehicle.
- We know that all vehicles will have some ID associated with them (registration plate number), a make and a model. If we consider the following vehicles: a truck, a car and a bicycle.
- All of these will have these characteristics associated with them. In addition, a truck will have a load capacity (maximum weight it can carry legally) and a number of axels under it.
- A car will have a number of doors associated with it and, the number of passengers it can legally carry.
- A bicycle will have the number of passengers it can carry and, the height of the saddle associated with it.

Inheritance: Example

- Even though all three vehicles are distinct, they have common features and we use inheritance as a means to represent this scenario and access these features.

Inheritance: Example

- In our example Vehicle will be a super class and Truck, Car and Bicycle will be subclasses.



Inheritance: Example

- In our example, the `Truck, Car` and `Bicycle` classes can keep the functionality and attributes of the Vehicle superclass and in addition have some more specific ones of their own.
- For example, in the `Car` class, it will have all the methods from Vehicle, namely, `getidNum(), getModel(), setModel(), toString()`.

- It can also have specific methods `like setNoDoors(), getnoPassengers(), setNoPassengers(),…`

- This means that although all classes have access to the methods and attributes in the `Vehicle` superclass we only have to write the code for them **once.**

Inheritance: Example 2

Let us look at another example of inheritance in java. Take an example of a `Dog` class where every dog has the following attributes:

- `Breed`
- `Colour`
- `Owner`
- `ChasesSquirrels`

In addition the `Dog` also has functionality to speak (growl and bark!).

Inheritance: Example 2

- We also have a `Cat` class which has the following attributes:
  - Breed
  - Colour
  - Owner
  - Works For Super Villain

This looks very similar to our `Dog` class.

- When we have shared attributes like this we can create a superclass with these and then inherit them in sub classes – our superclass will be `Animal`

Inheritance: Example 2

- Sample code to represent the Animal class is:

```
public class Animal
  {
      // Attributes
      private String breed;
      private String colour;
      private String owner;
      public Animal(String breed, String colour, String owner)
       {
            this.breed = breed;
            this.colour = colour;
            this.owner = owner;
       }

      public String getOwner()
       {
            return owner;
       }

      public void speak()
      {
         System.out.println("Grrr! Argh!");
      }
 }
```

Inheritance: Example 2

- This `Animal` class could be used to represent both a `Dog` and a `Cat`.
- However our `Dog` class has a `chaseSquirrels` attribute and our `Cat` class has a `worksForSuperVillain` attribute.
- So how do we inherit the shared traits?
- We use the Java keyword `extends`.
- This allows us to extend the design of the parent (super) class by enabling us to inherit all its attributes and functionality.

Inheritance: Extending a Class

- Our `Dog` class will now look like:

```
public Dog()
  {
        // Call the parent default constructor
        super();
        // Set attributes for Dog object
        this.chasesSquirrels = false;
}
```

Inheritance: Extending a Class

- The general `Dog` constructor might look like:

```
public Dog(boolean chasesSquirrels, String breed, String colour,
String owner)
 {
       // Call the parent general constructor
       super(breed, colour, owner);
       // Set attributes for Dog object
       this.chasesSquirrels = false;
 }
```

Inheritance: Extending a Class

- We can do something similar for the `Cat` class.

```
public class Cat extends Animal
{
        // Attributes
        boolean worksForSuperVillain;

   public Cat()
    {
        super();
        this.worksForSuperVillain = true;
    }

   public  Cat(String  breed,  String  colour,  String  owner,  boolean
   worksForSuperVillain)
    {
        // Call the parent general constructor
        super(breed, colour, owner);
        // Set attributes for Cat object
        this.worksForSuperVillain = worksForSuperVillain;
    }
}
```

Inheritance: Extending a Class

- We have now created a superclass, `Animal`, and two sub classes, `Dog` and `Cat`.
- We now need to create a `main()` method to allow us create instances of each of these, modify their attributes and print the values stores in the attributes.

```
public static void main(String args[])
{
        //Create an Animal object


        //Create a dog called fido
        Dog fido = new Dog();
        System.out.println("Fido chases Squirrels"+fido.getChasesSquirrels());
        //Create a cat

        Cat myCat = new Cat("Toby","black","Mary Jones",true);
        System.out.println("My Cat's Breed"+myCat.getBreed());
}
```

Inheritance: Extending a Class

- In the previous example we are call the `getChasesSquirrels()` method from the subclass `Dog`
- We call the `getBreed()` method from the superclass `Animal`.
- Inheritance **allows this**.
- We can override the behaviour of a super class method and add our own functionality on top of whatever is currently provided.
- We can even replace existing super class functionality with entirely new sub class behaviour.

Inheritance: Extending a Class

- To override a super class method and change its content we simply re-code the same method in the sub class.

- However we must maintain the same (or similar) method signature.

Inheritance: Extending a Class

For example, the method `speak()` in `Animal` looks like:

```
public void speak()
  {
        System.out.println("Grrr! Argh!");
  }
```

Inheritance: Extending a Class

In our subclass `Dog` we can write the same method signature but with different method content. For example: :

```
public void speak()
  {
        System.out.println("Woof! Woof!");
  }
```

Inheritance: Extending a Class

- If we do override a method in the superclass that the access modifiers for the overridden method cannot be more restrictive that the method overriding it.
- Now when we call the `speak()` method `in main()` for an `Animal` or a `Dog`, it will do different things:

```
public static void main(String[] args)
 {
       Animal a = new Animal();
       Dog d = new Dog();
       a.speak(); // prints "Grrr! Argh!"
       d.speak(); // prints "Woof! Woof!"
 }
```

Inheritance: Vehicle Example

- We will now look at writing a `Car` class which inherits from the `Vehicle` class.

- Firstly let us construct the `Vehicle` class:

```
//Superclass Vehicle

public class Vehicle
{
    // instance variables
    private int  numberOfWheels;
    private String owner;

    public Vehicle(int numberOfWheels, String owner)
     {
             this.numberOfWheels = numberOfWheels;
             this.owner = owner;
             System.out.println("Vehicle constructor called");
     }
    public Vehicle()
     {
             numberOfWheels = 4;
             owner = "<undefined>";
             System.out.println("Vehicle constructor called");
     }

    public void replaceAll()
     {
             System.out.println("We need " + numberOfWheels + " tyres to
             replace them all");
     }

     public int getNoWheels()
     {
             return numberOfWheels;
     }

     public String getOwner()
       {
             return owner;
       }

    public String toString()
     {
             return "This data for this Vehicle: Wheels: " + numberOfWheels
             +" owner: " + owner;
     }

}
```

Inheritance: Vehicle Example

Our `Vehicle` superclass has two instance variables which can be passed to the subclasses in addition to all of the methods within the class.

Inheritance: Vehicle Example

- Our `Car` class might look something like:

```
public class Car extends Vehicle
{
    // instance variables - replace the example below with your own
    private int noPassengers;

    /**
     * Constructor for objects of class Car
     */
    public Car(int num, String owner, int noPassengers)
    {
        super(num,owner);
        this.noPassengers = noPassengers;
        System.out.println("Car constructor called");
    }
    public Car(){

        super(); // no need for this as it is implicitly called
        noPassengers = 0;
        System.out.println("Car default constructor called");
    }
    public int getPassengers()
    {
       return noPassengers;
    }

    public String toString()
    {
       return "This data for this Car: Wheels: " + getNoWheels() +"  owner: "
       + getOwner() + "noPassengers: " + this.getPassengers();
    }

}
```

Inheritance: Vehicle Example

- Note how we use the `extends` keyword to infer inheritance.
- Also notice how each of the Car constructors is calling a different constructor from the `Vehicle` class using the `super()` methods.
- The final thing to do is to test it all. We will use a test file to bring it all together:

```
/**
 * Write a description of class Application here.
*/
public class Application
{
    public static void main(String [] args)
    {

            Car myCar = new Car();
            Car myNewCar = new Car(1562, "Ciara", 5);
            myCar.replaceAll();

            System.out.println("myCar data" + myCar.toString());
            System.out.println("myNewCar data" + myNewCar.toString());
    }
}
```

Inheritance: Vehicle Example

```
/**
 * Write a description of class Application here.
*/
public class Application
{
    public static void main(String [] args)
    {

            Car myCar = new Car();
            Car myNewCar = new Car(1562, "Ciara", 5);
            myCar.replaceAll();

            System.out.println("myCar data" + myCar.toString());
            System.out.println("myNewCar data" + myNewCar.toString());
    }
}
```

Inheritance: Summary

- Java allows us to reuse class definitions and **extend** functionality by allowing one class (the child or subclass) to **inherit** from another class (the parent or superclass) - This is known as **inheritance**
- All attributes and methods of the superclass are available in each subclass when inheritance is used
- The keyword `extends` allows one class to inherit from another
- In order to call the super class constructor we use the keyword `super()`
- We can override the behaviour of a super class method and add our own functionality on top of whatever is currently provided.