CS-1092
Welcome Back!

CS1092 : Inheritance in Java
(#1 of 4)

Gavin Brown

Kilburn Building, rm 2.81

gavin.brown@cs.man.ac.uk

Inheritance week

Today

The basics of inheritance

How to make use of it

Then… *Wednesday 11am*

Thursday 3pm

Friday 3pm – case study on using inheritance

It's 2010, and you just got headhunted by Porsche…

- Write a piece of code to control the intelligent auto-driving system.
  - *Methods for accelerating, turning, etc*
  - *Obey the speed limit*
  - *Variables to hold data, like current speed, speed limit, etc.*

```
public class Porsche {
   private double currSpeed = 0;
   private double nationalSpeedLimit = 100.0;

   public void accelerate() {
      if(currSpeed<nationalSpeedLimit)
          currSpeed = currSpeed + 1.0;
   }

   //more methods (turn, brake, etc)
}
```

```
public class Porsche {
   private double currSpeed = 0;
   private double nationalSpeedLimit = 100.0;

   public void accelerate() {
      if(currSpeed<nationalSpeedLimit)
          currSpeed = currSpeed + 1.0;
   }

   public double getSpeed() {
      return currSpeed;
   }
}
```

```
Porsche myPorsche = new Porsche();
Porsche theCar = new Porsche();

myPorsche.accelerate();
myPorsche.accelerate();
myPorsche.accelerate();
theCar.accelerate();

System.out.println( “Speed A: ” + myPorsche.getSpeed() );
System.out.println( “Speed B: ” + theCar.getSpeed() );
```

Porsche v2.0 is due for release!

```
public class Porsche {
   private double currSpeed = 0;
   private double nationalSpeedLimit = 100;
   private double percentBoost;
   private double versionNumber = 2.0;

   public void accelerate() {
      if(currSpeed<nationalSpeedLimit) {
         if (versionNumber == 2.0)
            currSpeed = currSpeed + (currSpeed*percentBoost)
         else
            currSpeed = currSpeed + 1.0;
      }
   }

   public void setTurboBoost( double perc ) {
      percentBoost = perc;
   }

   //more methods (turn, brake, etc)}
}
```

New feature:

"Turbo-boost"

Increases speed by given percentage

(added advantage that you can break the speed limit ;)

A smarter way… "overriding" the method

```
public class PorscheVersionTwo extends Porsche {
   private double percentBoost;

   public void accelerate() {
       currSpeed = currSpeed + (currSpeed*percentBoost);
   }

   public void setTurboBoost( double perc ) {
      percentBoost = perc;
   }
}
```

- *The important bit is "extends Porsche" – that makes inheritance happen.*
- *This class INHERITS all the methods/variables of the superclass.*

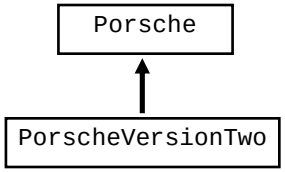
  …as if they were copied down into the subclass, automatically.
- *The accelerate method here OVERRIDES the one from the superclass.*
- *The setTurboBoost method ADDS EXTRA FUNCTIONALITY.*

The class we have written has the header:

```
public class PorscheVersionTwo extends Porsche
```

This is now called a subclass of the Porsche class.
The '`extends Porsche`' part makes sure that Java recognises it as so.

The Porsche class is said to be the *superclass* of the PorscheVersionTwo class.



Super-class

Sub-class

One little bug with that…

The PorscheVersionTwo we wrote WILL NOT COMPILE.

```
> javac PorscheVersionTwo.java
PorscheVersionTwo.java:5: currSpeed has private access in Porsche
       currSpeed = currSpeed + (currSpeed*percentBoost);
       ^
```

Why…. and how to fix it…

```
public class Porsche {
   private double currSpeed = 0;
. . .
```

"private" variables are INVISIBLE to the subclass.

```
public class Porsche {
   protected double currSpeed = 0;
. . .
```

Fix: make it "protected" instead.

One little bug with that… (fix #2)

Alternatively (and preferably) …

```
public class Porsche {
   private double currSpeed = 0;
. . .
```

```
public class Porsche {
   private double currSpeed = 0;

   public double getSpeed() {
       return currSpeed;
   }

. . .
```

Define an accessor method

In the examples we are about to see, both these ways are used, but this is the preferred way of dealing with private variables.

Accessing the original method:

```
public class PorscheVersionTwo extends Porsche {
   private double percentBoost;
   private boolean boostActive = false;

   public void accelerate() {
     if(!boostActive)
        super.accelerate();
     else
        currSpeed = currSpeed + (currSpeed*percentBoost)
   }

   public void setTurboBoost( double perc ) {
      boostActive = true;
      percentBoost = perc;
   }
}
```

Call the accelerate() method in the superclass

Overriding variables (“variable shadowing”)

(not so important or widely used as overriding methods)

```
public class PorscheVersionTwo extends Porsche
{
  private double nationalSpeedLimit;

  public void printSpeedLimits()
  {
    System.out.println(“Porsche v1.0 : “+super.nationalSpeedLimit)
    System.out.println(“Porsche v2.0 : “+nationalSpeedLimit )
  }
…
```

Access the one in the superclass

Access the one in this class

How about a class to control a clock? And a subclass for it…

```
public class Clock
{
   protected Time theTime;

   public update()
   {
     theTime.increment();
   }

   //other methods…
}
```

```
public class AlarmClock extends Clock
{
    private Time alarmTime;

    public update()
    {
      super.update();
      if (theTime.equals(alarmTime))
         System.out.println(“Beeeep!”);
    }
}
```

The update() method is overridden by the subclass, but it calls the original method using the “super” keyword we discussed.

Another example…

```
public class Bicycle
{
	private int numberOfGears;
	private int numberOfWheels;

	public void turn( double degrees )
	{
		//code for turning corners
	}

	// more methods (including accessors)
}
```

```
public class MountainBike extends Bicycle
{
	private double suspensionRatio;

	// more methods specific to Mountain bikes
}
```

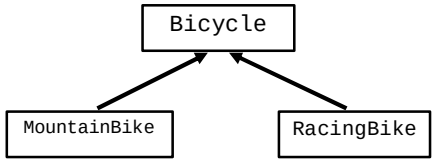
Several subclasses…

```
public class MountainBike extends Bicycle
{
	private double suspensionRatio;

	// more methods
}
```

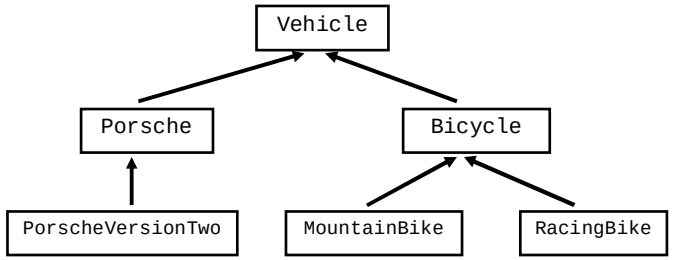
```
public class RacingBike extends Bicycle
{
	private String sponsorName;

	// more methods
}
```



We now have one superclass and two subclasses. The superclass holds the information that the two subclasses **have in common**– eg. the variables numberOfGears and numberOfWheels.

Class hierarchies



Q. What would the class headers be?

Class hierarchies

Q. What would the class headers be?

```
public class Vehicle

public class Porsche extends _________
public class PorscheVersionTwo extends _________

public class Bicycle extends ________
public class RacingBike extends __________
public class MountainBike extends ___________
```

Questions – 5 minutes…

1. What variables/methods would be in the Vehicle class?

2. How would you change the hierarchy above to account for different types of cars? Imagine Fords, Ferraris, and Rovers being included. How about different models of car - Ford Fiesta, Ford Escot, Ford Orion, etc.?

3. Think up another example where a hierarchy of classes might be necessary: write out the class headers, and the names of any variables/methods each may have (don't worry about the implementation of the methods).

4. (HARD ONE) It makes sense to have a "Porsche" object, and also a "MountainBike" object. Does it make sense to have a "Vehicle" object? Do your own reading on inheritance TONIGHT and see if you can find a sensible answer to this question.

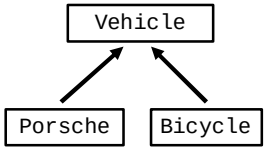
Inheriting the 'type' of a superclass

Look at the following statement:

```
int myNumber=42;
```

That is a variable – what type is it?
How about this one, what type is `myCar` ?

```
Porsche myCar = new Porsche();
```



If you said "Porsche", you're correct. If you said "Vehicle", you're also correct. Just like variables and methods are inherited, so is the 'type' of any superclasses. Thus the variable myCar in fact now has TWO types: Porsche, and Vehicle.

The PorscheVersionTwo class has THREE types:
`PorscheVersionTwo, Porsche, and Vehicle`!

What does this mean for us? Think carefully about how you create an object…

```
Porsche myCar = new Porsche();
Camera myCamera = new Camera();
```

A general code 'template' for making an object this way is thus:

```
<type> <name> = new <type>();
```

So since the myCar object now has TWO types, we could write:

```
Vehicle myCar = new Porsche();
```

But… we cannot do it the other way around

```
Porsche myCar = new Vehicle();    // ERROR!
```

To remember which way round this goes, just say this in your head:

1. ***"all Porsches are Vehicles"***
2. ***"all Vehicles are Porsches"***

…then decide which is true. Since there are obviously some Vehicles that are not Porsches, you cannot assign an arbitrary Vehicle object to a Porsche type variable.

Or remember this….

```
Vehicle myVehicle     = new Porsche();
<general-thing>       = <more-specific-thing>  //ok :)


Porsche myCar         = new Vehicle();
<more-specific-thing> = <general-thing>        //No!
```

Inheriting the 'type' of your superclass is known as 'polymorphism'.

And we'll see more of that tomorrow…

Concepts covered today:

1. Subclasses and superclasses
2. Inheriting methods
3. Inheriting variables
4. Adding new methods/variables
5. Using the "super" keyword to access the original method/variable
6. Class hierarchies
7. Inheriting the type ("polymorphism")

Tomorrow :

1. Superclass… superconstructor
2. More polymorphism – and making good use of it
3. "Casting" objects
4. Checking what type an object is at runtime
5. "abstract" classes