CS1092 : Inheritance in Java (#2 of 4)

Gavin Brown

Kilburn Building, rm 2.81

gavin.brown@cs.man.ac.uk

www.cs.man.ac.uk/~gbrown/teaching/java/

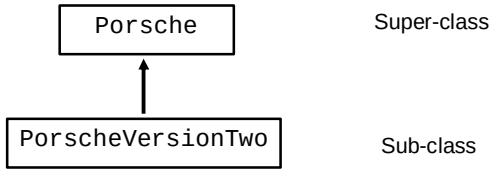
REMINDER OF YESTERDAY

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.



REMINDER OF YESTERDAY

```
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.*

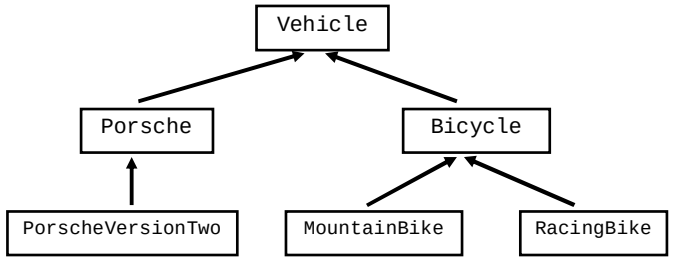
REMINDER OF YESTERDAY

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

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

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

       // more methods
}
```

REMINDER OF YESTERDAY

Class Hierarchies



The subclasses inherit the 'type' of their superclass.
Q. What type is MountainBike?

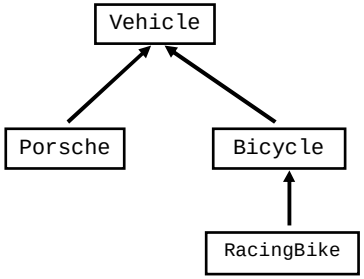
This is called 'polymorphism'.

Today

1. Design – variables in the hierarchy?
2. More polymorphism – making use of it
3. Casting to subclasses
4. Checking the type of an object at runtime
5. Superclass… superconstructor
6. "abstract" classes

Design : Where do variables go?

```
class Vehicle
   int numberOfWheels

class Porsche extends Vehicle
   String numberPlate
   double currSpeed
   double nationalSpeedLimit

class Bicycle extends Vehicle
   int numberOfGears

class RacingBike extends Bicycle
   String sponsorName
```



Variables in COMMON go at the top of the hierarchy.

Variables specific to a class go in that class.

The same applies for methods.

Making use of polymorphism

Polymorphism (inheriting the 'type') makes this code possible:

```
Vehicle v1 = new Porsche();
Vehicle v2 = new Bicycle();
Vehicle v3 = new Plane();

System.out.println(v1.numberOfWheels)
System.out.println(v2.numberOfWheels)
System.out.println(v3.numberOfWheels)
```

Let's run that code:

```
> java test
4
2
6
```

Making use of polymorphism

Or even more useful, an array of Vehicle objects:

```
Vehicle [] vehicleList = new Vehicle[4];

vehicleList[0] = new Porsche();
vehicleList[1] = new Bicycle();
vehicleList[2] = new Plane();
vehicleList[3] = new Porsche();

for (int i=0; i<4; i++)
    System.out.println(vehicleList[i].numberOfWheels)
```

Let's run that code:

```
> java test
4
2
6
4
```

Be careful with types…

```
vehicleList[3] = new Porsche();

Vehicle v = vehicleList[3];
System.out.println( v.numberOfWheels );  //ok
System.out.println( v.numberPlate );     //ERROR
```

(because *numberPlate* is part of the **Porsche** class, not the **Vehicle** class…)

```
> javac test.java

test.java:12: cannot resolve symbol
symbol  : variable numberPlate
location: class Vehicle
        System.out.println( v.numberPlate );
                                  ^
1 error
```

Be careful with types : casting to the subclass

The object has to be cast to the subclass first:

```
Porsche p = (Porsche)vehicleList[3];
System.out.println( p.numberPlate );  //ok! :)
```

The word 'Porsche' in brackets tells Java which type we want to cast to.

```
> javac test.java
> java test
<NO-NUMBERPLATE-REGISTERED>
>
```

Checking the type of an object

```
Vehicle [] vehicleList = new Vehicle[4];

vehicleList[0] = new Porsche();
vehicleList[1] = new Bicycle();
vehicleList[2] = new Bicycle();
vehicleList[3] = new Porsche();

for (int i=0; i<4; i++) {
   System.out.print("In your array position "+i+" is");

   if (vehicleList[i] instanceof Porsche) {
      System.out.println(" a fast car!");
   }

   if (vehicleList[i] instanceof Bicycle) {
      System.out.println(" my bike.");
   }
}
```

Checking the type of an object

```
> javac test.java
> java test
In your array position 0 is a fast car!
In your array position 1 is my bike.
In your array position 2 is my bike.
In your array position 3 is a fast car!
>
```

Superclass…superconstructor

```
> java test
<NO-NUMBERPLATE-REGISTERED>
```

```
public class Porsche
{
    public String numberPlate;

    public Porsche( String plate )
    {
       numberPlate = plate;
    }
…
```

```
public class PorscheVersionTwo extends Porsche
{
    public PorscheVersionTwo( String plate )
    {
       super(plate);
       numberPlate = "$$ "+numberPlate+ " $$"
    }
…
```

Superclass…superconstructor

```
PorscheVersionTwo p2 = new PorscheVersionTwo(“cdc 9922”);
System.out.println(p2.numberPlate);
```

```
> java test
$$-cdc-9922-$$
```

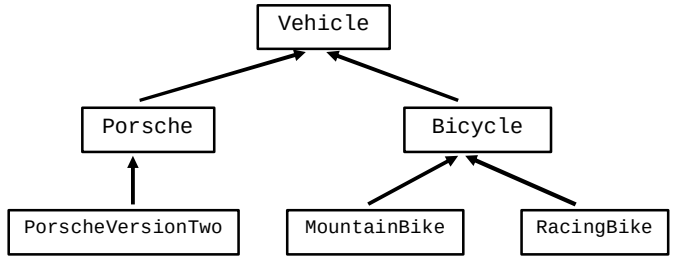
Call to superconstructor MUST be first!

```
public class PorscheVersionTwo extends Porsche
{
    public PorscheVersionTwo( String plate )
    {
       super(plate);
       numberPlate = ”$$-” +numberPlate+ ”-$$”
    }
…
```

5 minutes…

How many wheels on a bike?

Q. It makes sense to have a "Porsche" object, and also a "MountainBike" object. Does it make sense to have a "Vehicle" object?




```
Porsche p = new Porsche();
RacingBike rb = new RacingBike();

Vehicle v = new Vehicle();    // ????
```

Technically correct, but what does it mean?

Abstract class: Vehicle

```
public abstract class Vehicle
{
       private int numberOfWheels;

       public int getNumWheels() {
           return numberOfWheels;
       }

       public abstract void turn();
}
```

Provides data and methods common to all Vehicles

And forces ALL subclasses to have a "turn()" method

Abstract classes : *Useful software engineering tool: when working in a team, write an abstract class and give it to a colleague to work from. You can provides some functionality, and impose some rules, like the above class making the rule that subclasses should have a turn() method.*

Extending from the abstract Vehicle class

```
public class Plane extends Vehicle
{
       private double wingspan;

       public void takeOff() {
          // more code
       }
}
```

```
> javac Plane.java
Plane.java:1: Plane should be declared abstract; it does
not define turn() in Vehicle
public class Plane extends Vehicle
       ^
1 error
```

Here, the compiler is telling us that the `Plane` should be made abstract, because we do not define `turn()`. In fact, we should just define the `turn()` method and all will be fine. The compiler gave the best advice it could, assuming that we wanted `Plane` to be abstract, and forgot to do so.

Today

1. Design – variables in the hierarchy
2. More polymorphism – making use of it
3. *Casting to subclasses*
4. Checking object type: *instanceof*
5. Superclass… superconstructor
6. “Abstract” classes

Tomorrow

1. *More on abstract classes*
2. *dynamic method binding*
3. *is-a versus has-a rules*
4. *‘final’ keyword to control inheritance*
5. *constructor ordering*