CT2106
Object Oriented Programming

Dr. Frank Glavin
Room 404, IT Building
Frank.Glavin@University*of*Galway.ie
School of Computer Science

University *of* Galway.ie

Today's topics

- Equal Method
- Java.lang.Object
- Checking type of a variable
- Casting
- Type hierarchy
- Understanding basic inheritance
- Overriding

Ollscoil na Gaillimhe
University of Galway

equals

Your equals method should look like this

```
/**
 * version 1 of an equals method
 * @param  object  a parameter of type Bicycle
 * @returns true or false
 */
public boolean equals(Bicycle bike){

    //TODO - we will write the body here
    return false; // default return value
}
```

Ollscoil na Gaillimhe
University of Galway

How to use it (even though it's incomplete)

We would use it the same way we used the equals method of the String class

E.g.

```
Bicycle bike1 = new Bicycle(4,20);
Bicycle bike2 = new Bicycle(2,10);

if(bike2.equals(bike1)){
    System.out.println("true");
}else{
    System.out.println("false");
}
```

Ollscoil na Gaillimhe
University of Galway

Bicycle Equality

- What would make two Bicycle objects equal in terms of value?
- **Their state**. i.e. their field values: speed and gear
- The same applies for every object

Ollscoil na Gaillimhe
University of Galway

Your task

In the section marked //TODO in the equals method write code so that

```
/**
 * version 1 of an equals method
 * @param  object  a parameter of type Bicycle
 * @returns true or false
 */
public boolean equals(Bicycle bike){

    //TODO - write code as described below
    //If the speed AND the gear values of this bicycle object are equal (==)
    //to the speed AND gear values of the input Bicycle
    //return true else return false

    return false; // default return value
}
```

Ollscoil na Gaillimhe
University of Galway

equals

Your equals method should look like this

```
/**
 * version 1 of an equals method
 * @param  bike  a parameter of type Bicycle
 * @return    true or false
 */
public boolean equals(Bicycle bike){
    if(speed == bike.getSpeed() && gear == bike.getGear()){
        return true;
    }   //no need for an else statement
    return false; // default return value
}
```

Ollscoil na Gaillimhe
University of Galway

Test in CodePad (or in a main method)

```
public static void main(String[] args)
{
    Bicycle bike1 = new Bicycle(4,20);
    Bicycle bike2 = new Bicycle(2,10);

    if(bike2.equals(bike1)){
        System.out.println("true");
    }else{
        System.out.println("false");
    }

}
```

Ollscoil na Gaillimhe
University of Galway

Tip

Java has a shorthand for writing if/else statements

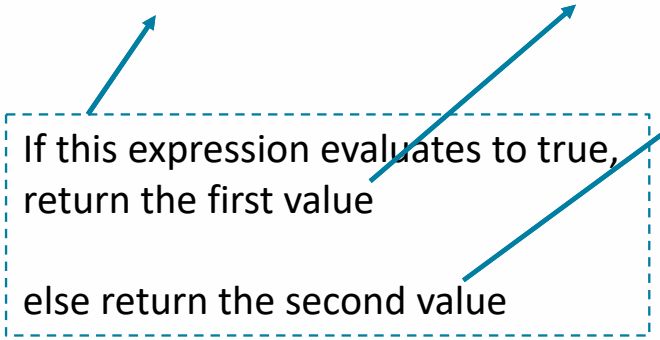
```
if(bike2.equals(bike1)){
        System.out.println("true");
}else{
        System.out.println("false");
}
```

Ollscoil na Gaillimhe
University of Galway

Tip: Ternary operator

It can be reduced to one line

```
System.out.println(bike2.equals(bike1)? "true”  :"false");
```



Ollscoil na Gaillimhe
University of Galway

Ok, so we cheated

- Our version of the equals method doesn't comply with the standard equals method expected by the Java language.
- It is nearly there but not quite
- To understand this fully we will encounter a fundamental concept in OOP **: inheritance**

Ollscoil na Gaillimhe
University of Galway

Correct version

- **All equals methods must have the following method signature**
- public boolean equals(Object object)

Ollscoil na Gaillimhe
University of Galway

V1 of equals method for Bicycle class

Your equals method should look like this

```
/**
 * version 1 of an equals method
 * @param  bike  a parameter of type Bicycle
 * @return    true or false
 */
public boolean equals(Bicycle bike){
    if(speed == bike.getSpeed() && gear == bike.getGear()){
        return true;
    }   //no need for an else statement
    return false; // default return value
}
```

Ollscoil na Gaillimhe
University of Galway

However

This version of the equals method **doesn't comply with the standard equals method expected by the Java language.**

Ollscoil na Gaillimhe
University of Galway

Correct version

The Java language has a standard signature for the equals method:

public boolean equals(java.lang.Object object)

Ollscoil na Gaillimhe
University of Galway

Version2 of equals method

```
37  /**
38   * version 2 of an equals method
39   * @param  object  a parameter of type Bicycle
40   * @returns true or false
41   */
42  public boolean equals(Object obj){
43
44      if(obj==null){
45         return false;
46      }
47
48      if (obj instanceof Bicycle){
49
50          Bicycle bike = (Bicycle)obj;
51
52          if(this.speed ==bike.getSpeed() && this.gear==bike.getGear()){
53              return true;
54          }
55
56      }
57      return false;
58  }
```

5 minute challenge

- Write down in pseudocode (or your own words) what you think this method is doing?
- Question to try to answer:
  - What is the meaning of **Object** in this piece of code?
  - Why is there a check for null?
  - What does the **instanceof** operator do?
  - What is happening in this line of code:
    - Bicycle bike = (Bicycle)obj;
- It may be helpful to contrast if with the previous version

Ollscoil na Gaillimhe
University of Galway

Checking for null

- Checking for a null value is very commonly done.
- It anticipates the program throwing a null pointer exception
- For example, the following code would throw a NullPointerException if we didn't check for null in the equals method

```
Bicycle bike1 = null;
Bicycle bike2 = new Bicycle(2,10);

System.out.println(bike2.equals(bike1)? "true" : "false");
```

Ollscoil na Gaillimhe
University of Galway

instanceof

instanceof is an operator that is used to determine if variable is pointing to an object with a particular type;

```
if (bike2 instanceof Bicycle){
    System.out.println("true");
}else{
     System.out.println("false");
}
```

Ollscoil na Gaillimhe
University of Galway

instanceof

instanceof is an operator that is used to determine if a variable is pointing to an object with a particular type;

```
System.out.println(bike2 instanceof Bicycle? "true" : "false");
```

From now on I will use this notation to represent if/else examples

Ollscoil na Gaillimhe
University of Galway

Object

```
public boolean equals(Object obj)
```

Q: What is the parameter Object obj?
A: obj is variable whose type is **java.lang.Object**

Q: What is java.lang.Object?
A: It is a class that provides the **most generic definition** of an object in Java

Ollscoil na Gaillimhe
University of Galway

Java.lang.Object

- java.lang.Object is the **parent class** of every class in Java
- A Bicycle object is a Bicycle object AND a java.lang.Object object
- A String object is a String object AND a java.lang.Object object

Ollscoil na Gaillimhe
University of Galway

Let’s test these ideas

- Open BlueJ
- Create a new Project
- Download the Bicycle class from Blackboard
- Right-click on the BlueJ workbench, select “Add Class from File”
- Add the Bicycle class you have downloaded
- Compile

Ollscoil na Gaillimhe
University of Galway

Open BlueJ

Open the project where you have the Bicycle class defined
Make sure CodePad is open

```
Bicycle bike = new Bicycle(2,12);
Bicycle bike2 = new Bicycle(3,15);
Object anObject = new Bicycle(3,15);
Object obj = "Hello World"; // String object
```

Variable `anObject` of type Object is pointing to a Bicycle object

This is allowed because the Bicycle object is a Bicycle type and an Object type

All Java objects are an Object type

Object type is short for java.lang.Object type

Ollscoil na Gaillimhe
University of Galway

Open BlueJ

Variable	Var type	Object	Object Type
bike	Bicycle	New Bicycle(2,12)	Bicycle
bike2	Bicycle	New Bicycle(3,15)	Bicycle
anObject	Object	New Bicycle(3,15)	Bicycle
obj	Object	“Hello World”	String

```
Bicycle bike = new Bicycle(2,12);
Bicycle bike2 = new Bicycle(3,15);
Object anObject = new Bicycle(3,15);
Object obj = "Hello World"; // String object
```

Variables of type Object can point to any type of Object

Ollscoil na Gaillimhe
University of Galway

Using instanceof to check object type

```
boolean isBike = anObject instanceof Bicycle; // is the object referenced a Bicycle
isBike
    true   (boolean)
boolean isString = anObject instanceof String; // is the object referenced a String object
isString
    false   (boolean)
 isString = obj instanceof String; // is the object referenced a String object
isString
    true   (boolean)
```

Ollscoil na Gaillimhe
University of Galway

Using the new equals method

```
boolean isEqual = bike2.equals(anObject);
isEqual
    true   (boolean)
 isEqual = bike2.equals(obj);
isEqual
    false   (boolean)
```

Ollscoil na Gaillimhe
University of Galway

Casting

- Here we cast (convert) a variable from a higher type (Object) to a lower type (Bicycle)
- We do the same for the String
- This is allowed as
  - `anObject` points to a Bicycle object – we've checked this using `instanceof`
  - `obj` points to a String object - we've checked this using `instanceof`

```
Bicycle bike3 = (Bicycle)anObject;
String str5 = (String)obj;
```

Ollscoil na Gaillimhe
University of Galway

Note that it is the variable type that is being converted **not the object**

```
Bicycle bike3 = (Bicycle)anObject;
String str5 = (String)obj;
```

Variable	Var type	Object	Object Type
anObject	~~Object~~	New Bicycle(3,15)	Bicycle
obj	~~Object~~	“Hello World”	String

Ollscoil na Gaillimhe
University of Galway

Java organises all its classes in a class hierarchy

Ollscoil na Gaillimhe
University of Galway

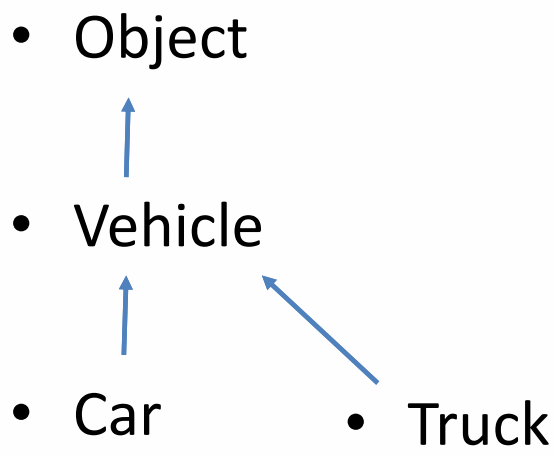
Consider The Real World…

- Object
- Vehicle
- Car

- In the most general sense, **a single car** is an object.
- But more specifically, it is a vehicle
- And more specifically still, it is a car

Ollscoil na Gaillimhe
University of Galway

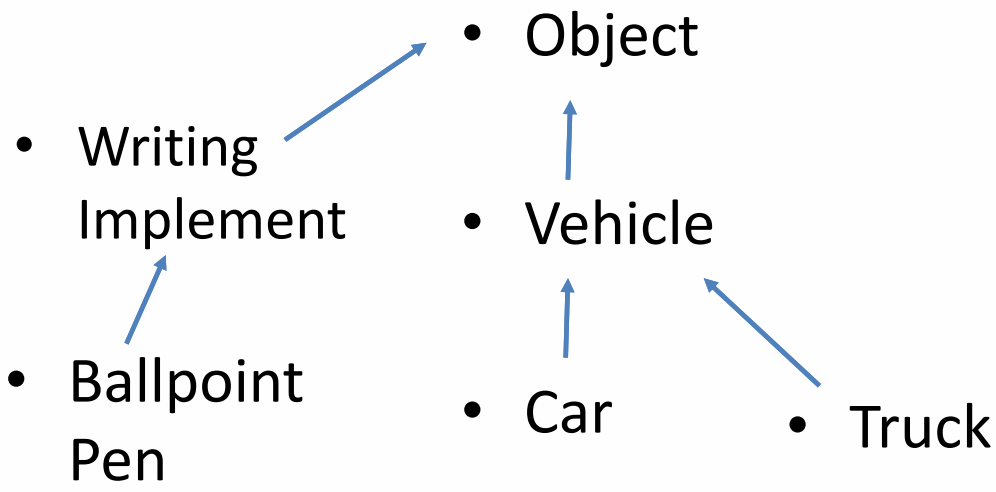
The Real World



- In the most general sense, a single truck is an object.
- But more specifically it is also a Vehicle
- And more specifically still it is a type of Truck

Ollscoil na Gaillimhe
University of Galway

The Real World

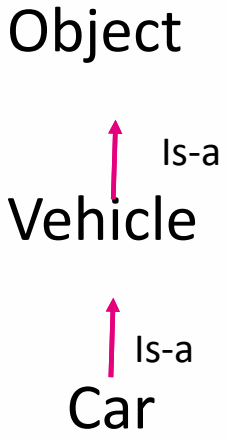


- In the most general sense, **a ballpoint pen** is an object.
- But more specifically it is a writing implement
- And more specifically still it is a ballpoint pen

Ollscoil na Gaillimhe
University of Galway

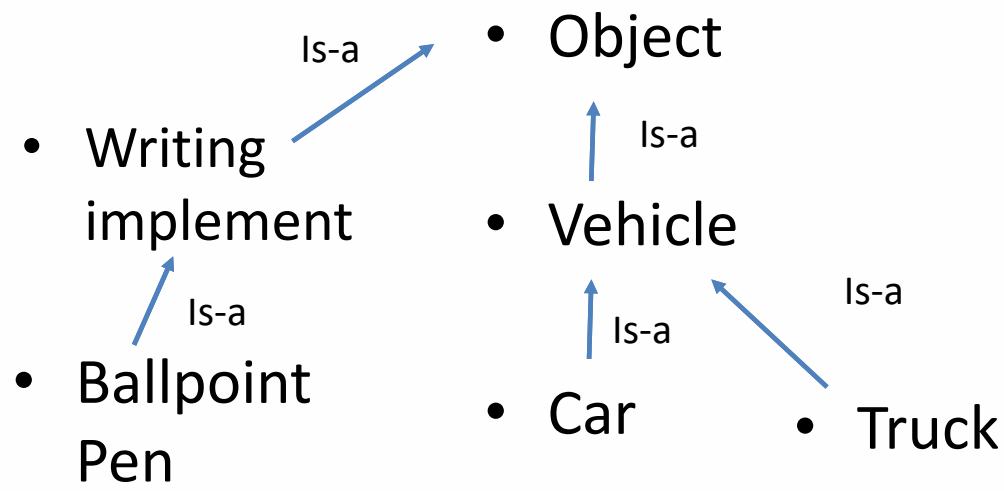
Is-a

- These relationships can be described as “is-a” relationships
- A car is-a vehicle; a vehicle is-a(n) object
- We can call the higher up types as **parent**s and the lower ones **children**
- Car **is a child** of Vehicle
- Vehicle **is a parent** of Car
- Object **is the parent** of Vehicle and Car

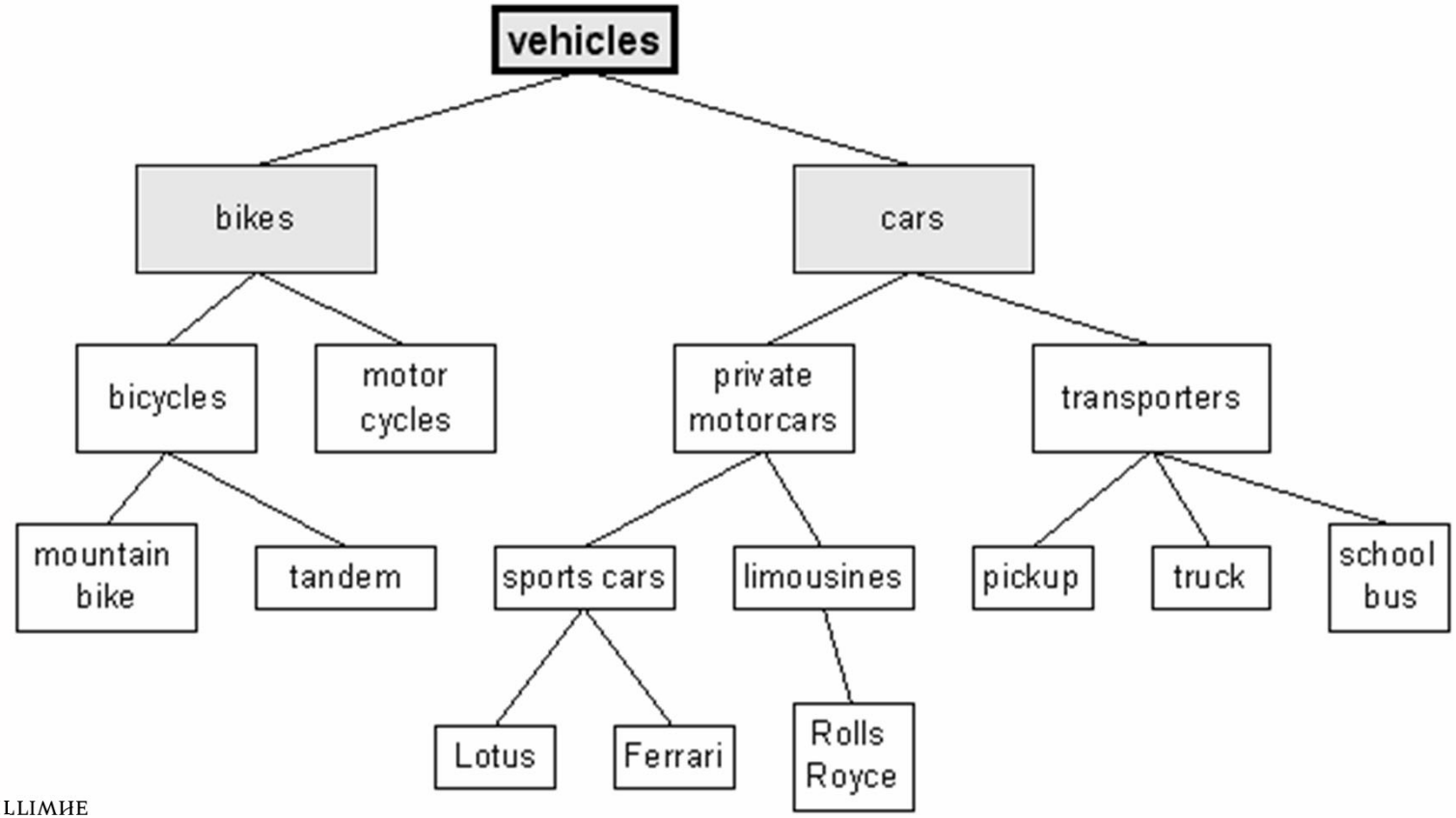


Ollscoil na Gaillimhe
University of Galway

The Real World



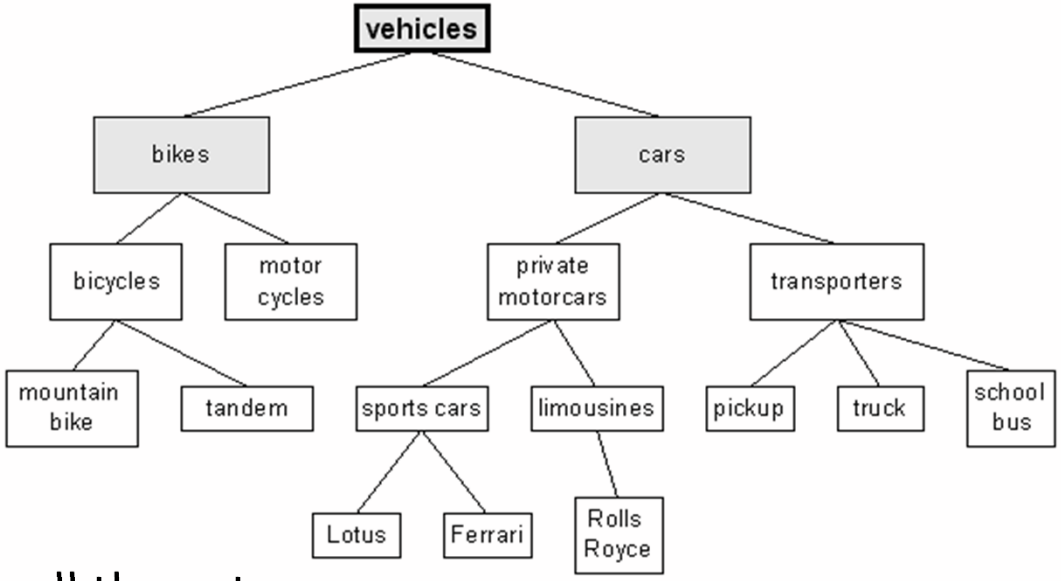
We see these concept hierarchies all over the non-Java world



Ollscoil na Gaillimhe
University of Galway

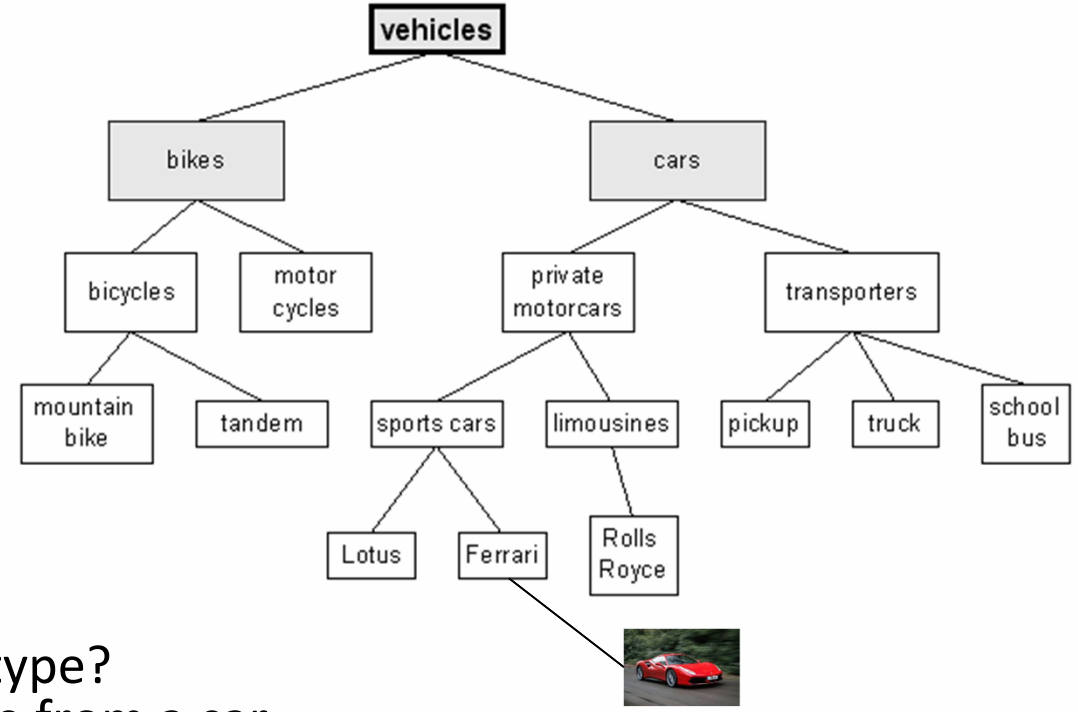
Ferrari 488

a. vehicle
b. car
c. private car
d. sports car
e. Ferrari



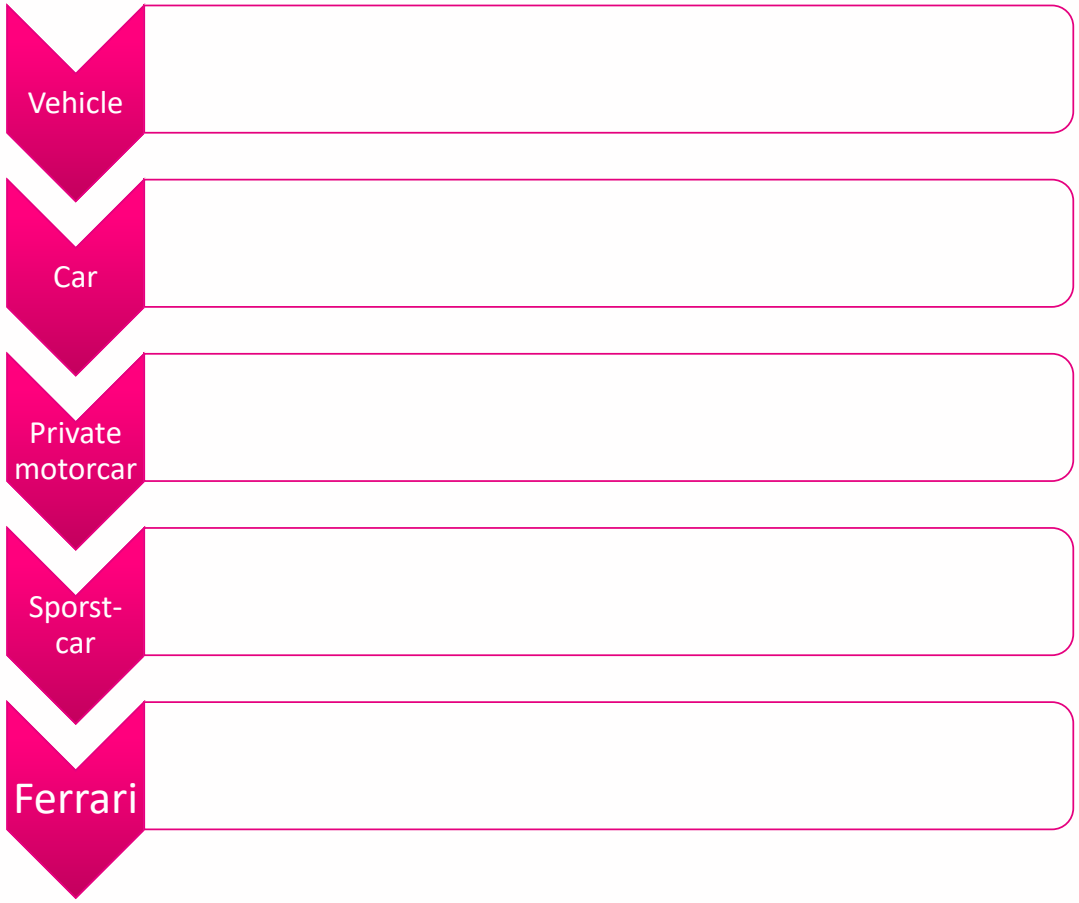
Ferrari 488 has **is-a** relationships with all these types
Its most specific type is Ferrari
Its most generic type is vehicle

a. vehicle
b. car
c. private car
d. sports car
e. Ferrari

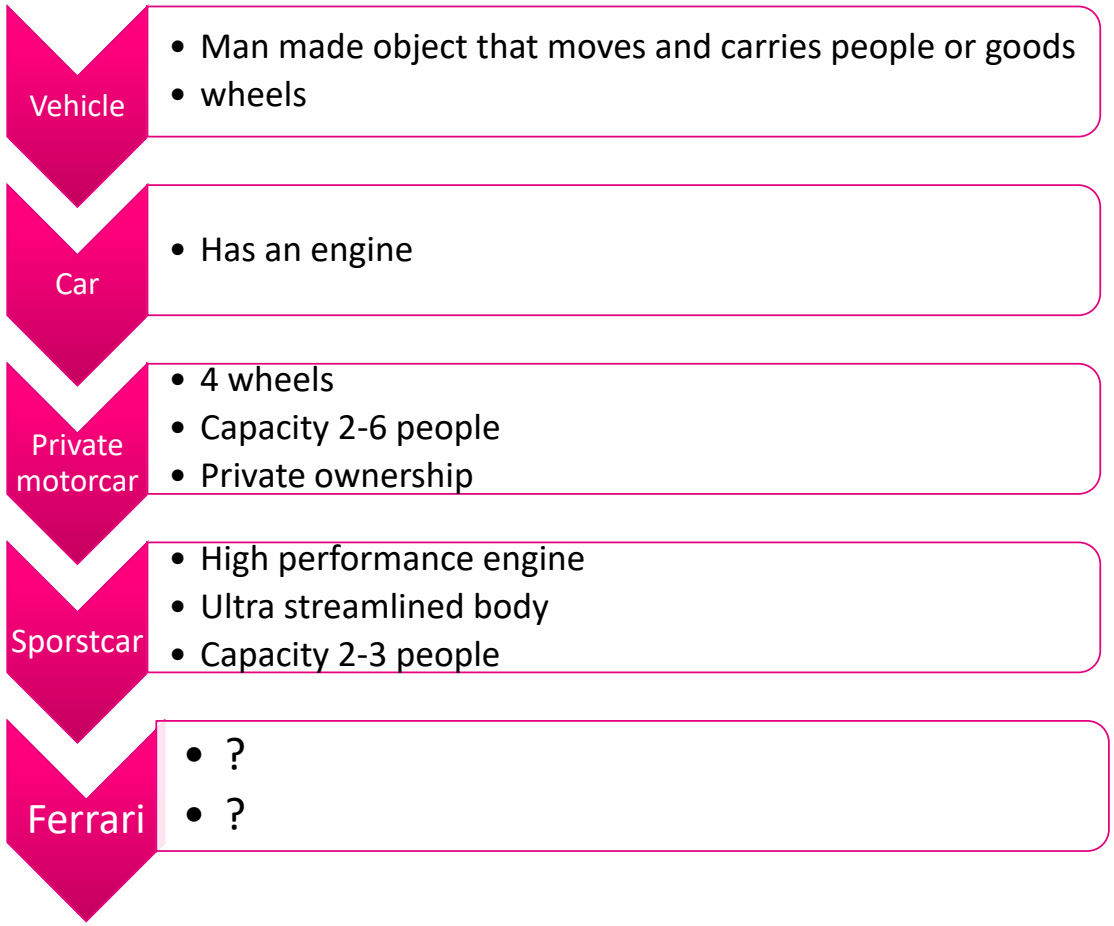


Question: what distinguishes each type?
E.g. a car from a sports car, a vehicle from a car

What are features and behaviours of each



What are features and behaviours of each:



Key idea in a class hierarchy

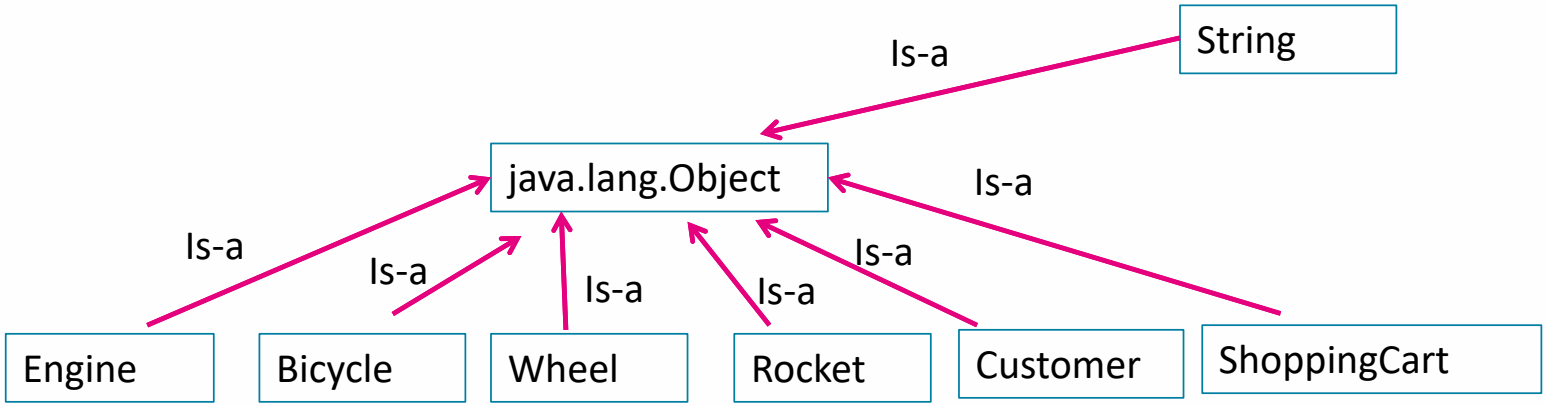
- The top of the hierarchy represents the **most generic attributes and behaviours**
- The bottom (the leaves) represent the **most specific attributes and behaviours**

- **Each level inherits and customises the attributes and behaviours from the level above it**

Ollscoil na Gaillimhe
University of Galway

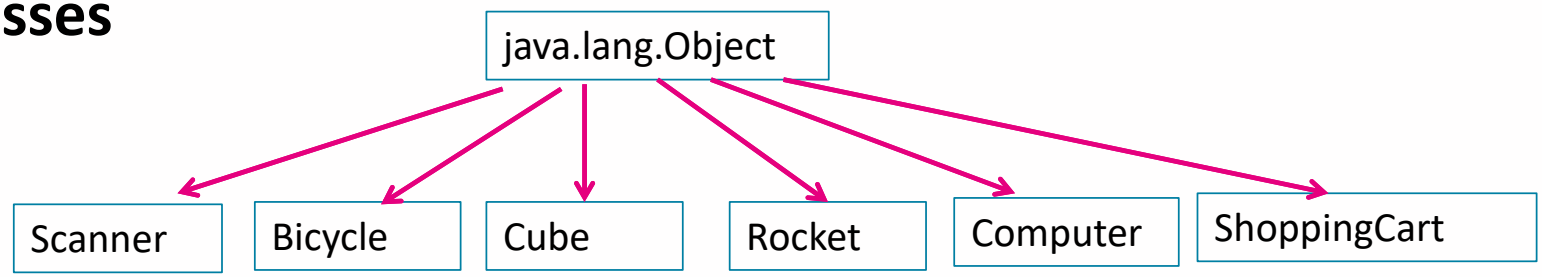
Back to Java

- Java.lang.Object is **THE superclass**, the parent of all classes in Java
- Every class in Java has java.lang.Object as its superclass (its parent)



Superclass

- In Java all child, or sub-classes **inherit** properties and methods from their **superclasses**



- All the classes shown above inherit (receive) methods from the superclass java.lang.Object
- Even though these methods may not be shown in the Class code – they are still available to objects of the Class

Ollscoil na Gaillimhe
University of Galway

OOP Inheritance

- The means by which objects automatically receive features (fields) and behaviours (methods) from their **superclasses**

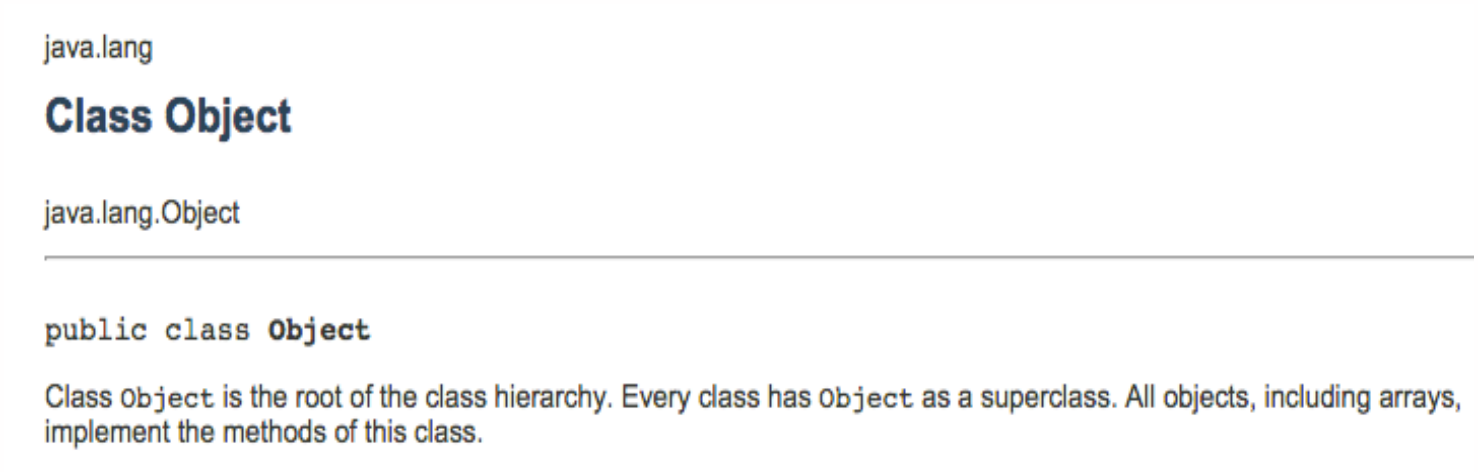
Ollscoil na Gaillimhe
University of Galway

Java.lang.Object

- **equals is one of the methods provided by java.lang.Object**
- Look up the java.lang.Object class definition on the Web:
- http://docs.oracle.com/javase/8/docs/api/java/lang/Object.html

Ollscoil na Gaillimhe
University of Galway

http://docs.oracle.com/javase/8/docs/api/java/lang/Object.html



java.lang

Class Object

java.lang.Object

```
public class Object
```

Class `Object` is the root of the class hierarchy. Every class has `Object` as a superclass. All objects, including arrays, implement the methods of this class.

Ollscoil na Gaillimhe
University of Galway

java.lang

Class Object

Methods

Modifier and Type	Method and Description
`protected Object`	**`clone()`** Creates and returns a copy of this object.
`boolean`	**`equals(Object obj)`** Indicates whether some other object is "equal to" this one.
`protected void`	**`finalize()`** Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
`Class<?>`	**`getClass()`** Returns the runtime class of this `Object`.
`int`	**`hashCode()`** Returns a hash code value for the object.
`void`	**`notify()`** Wakes up a single thread that is waiting on this object's monitor.
`void`	**`notifyAll()`** Wakes up all threads that are waiting on this object's monitor.
`String`	**`toString()`** Returns a string representation of the object.
`void`	**`wait()`** Causes the current thread to wait until another thread invokes the **`notify()`** method or the **`notifyAll()`** method for this object.
`void`	**`wait(long timeout)`** Causes the current thread to wait until either another thread invokes the **`notify()`** method or the **`notifyAll()`** method for this object, or a specified amount of time has elapsed.
`void`	**`wait(long timeout, int nanos)`** Causes the current thread to wait until another thread invokes the **`notify()`** method or the **`notifyAll()`** method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.

Ollscoil na Gaillimhe
University of Galway

Generic methods

- All the methods provided by java.lang.Object are *generic*
- They only relate *to* java.lang.Object but not to the subclasses
- When a subclass inherits these methods, it needs to customise them
- This is why we have to write own version of equals() for the Bicycle class
- **Overriding:** when you write your own version of a method that you have inherited from a superclass

Ollscoil na Gaillimhe
University of Galway

Overriding

When overriding a method, you must keep every part of the method signature the same
You only change the code in method body

```
37  /**
38   * version 2 of an equals method
39   * @param  object  a parameter of type Bicycle
40   * @returns true or false
41   */
42  public boolean equals(Object obj){
43
44      if(obj==null){
45         return false;
46      }
47
48      if (obj instanceof Bicycle){
49
50          Bicycle bike = (Bicycle)obj;
51
52          if(this.speed ==bike.getSpeed() && this.gear==bike.getGear()){
53              return true;
54          }
55
56      }
57      return false;
58  }
```

Method signature

Method Body

Ollscoil na Gaillimhe
University of Galway

Overriding rules

- **Overriding** a method means creating a specific version of a method inherited from a parent (superclass) class
- The only rule is that you keep the **signature** of the method the same
- Its name (e.g. equals)
- Its parameter types and order
- Its access level (e.g. public, protected)
- Its return type (e.g. boolean)

Ollscoil na Gaillimhe
University of Galway

This is called an **Annotation**. Your code will compile and run without it. But is considered good practice to annotate the methods that are *overridden* versions inherited from the superclass

```
51  /**
52   * version 2 of an equals method
53   * @param  object a parameter of type Bicycle
54   * @returns true or false
55   */
56  @Override
57  public boolean equals(Object obj){
58
59      if(obj==null){
60          return false;
61      }
62
63      if (obj instanceof Bicycle){
64
65          Bicycle bike = (Bicycle)obj;
66
67          if(this.speed ==bike.getSpeed() && this.gear==bike.getGear()){
68              return true;
69          }
70
71      }
72      return false;
73  }
```

Ollscoil na Gaillimhe
University of Galway

After this week's work, you should know how to:

- Write an equals method with the correct signature for any object
- Explain what java.lang.Object is
- Explain why you may need to check if a reference variable points to null
- Write an expression that uses **instanceof** to check the type (class) of an object
- Explain the concept of superclass and subclass
- Explain that the type of a reference variable can differ from the type of the object it points to
- Following from the last point, explain that the type of the variable can only be of a class (type) higher in the class hierarchy that the object that it points to
- Object obj = new Bicycle(2,14);
- Explain the basic idea of inheritance: properties and methods are received (inherited) from the superclass(es)
- Explain and demonstrate that subclasses can specialise and create their own versions of methods inherited from the superclass. This is called **overriding.** Inherited methods that are customised by the subclass are said to be "overridden"

Ollscoil na Gaillimhe
University of Galway