Programming 1

Introduction to Java
Lecture #3: Introduction to object orientation
(or “everything you’ve learned is a lie”)

The story so far…

- Up until now we've been approaching programming from a *procedural* perspective
  - Our programs have been a set of steps – a *procedure* – which solves a given problem or task
  - If a person (or a computer) starts at the beginning, and follow the procedure step by step they'll end up with the result or answer to the problem

Recap: The Java Programming Language

- Java is a general purpose language
- Java is **platform independent**
- Java is an **object-oriented** language
- There are different ‘editions’ of Java – we are using Java SE
- Those of you in the "A" occurrence of the module will get to use Java EE next teaching block…

What is Object Oriented Programming?

- A software design method based around writing programs that model real world* objects
- Real world objects have
  - behaviour (what it *does*)
  - state (its *characteristics*, what it *has*)
  - identity (what *makes it different* from others)
- Real world objects interact with other objects
  - In an object oriented program, the interactions between objects form the "glue" of the program…

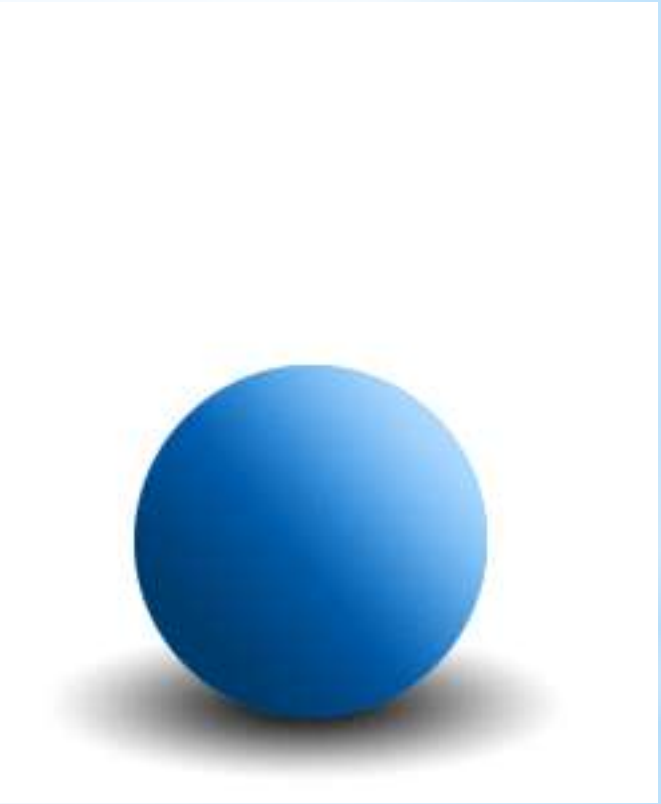
* although often the object might be an abstract thing…

Our example is a load of balls

- A ball has
  - diameter
  - colour

Our example is a load of balls

- A ball can
  - bounce
  - roll



Our example is a load of balls

- There are different types of balls

- They are different because of their **attributes** - e.g. their colour and diameter
- They might bounce and roll differently (maybe as a result of their different diameter)

The terminology of OO

- Think of a *class* as being like the blueprint for an object
  - If we have a class called **Ball**, it will contain the code that describes the characteristics of each and every ball
- The *attributes* of a class describe how individual objects of that class differ from each other
  - Consider a class called **Ball**, with attributes **diameter** and **colour**...
    - a cricket ball has a different diameter and colour to a tennis ball
    - A tennis ball is an instance of the class ball, so is a cricket ball, but they are DIFFERENT instances of the same class
- The *methods* of a class describe what individual objects of that class can do – the actions they can take
  - these methods might different from instance to instance based on their attributes

Drawing back the curtain

- Everything in Java is defined as a **class**
  - Again: think of a class as being like the blueprint for an object. It does not detail a specific single one of the thing you're describing – only a general definition
  - Once we have this blueprint, we can use it to create **instances** of our classes – or **objects**
- Up until now NoobLab has been hiding the **class declaration** and the **main method** from you
- This has meant you can use Java in a procedural way – but Java is NOT a procedural language…

Remember The simplest Java program: Hello World!

NoobLab takes this as read in the early exercises

```
public class SomeJavaCode
{
    public static void main (String[] args)
    {
        System.out.println("Hello, World!");
    }
}
```

and this

Back to our balls

- A Java class to define a ball might look like this

```
public class Ball
{
   public double diameter;
   public String colour;

   public void bounce()
   {
      System.out.println("Boing");
   }

   public void roll()
   {
      System.out.println("Whee");
   }
}
```

Back to our balls

- A Java class to define a ball might look like this

```
public class Ball
{
    public double diameter;
    public String colour;

    public void bounce()
    {
        System.out.println("Boing");
    }

    public void roll()
    {
        System.out.println("Whee");
    }
}
```

Back to our balls

- A Java class to define a ball might look like this

```
public class Ball
{
   public double diameter;
   public String colour;

   public void bounce()
   {
      System.out.println("Boing");
   }

   public void roll()
   {
      System.out.println("Whee");
   }
}
```

Attributes

More on our balls

- Some things to note
  - Class names should be capitalised (remember the conventions of Java!)
  - Curly brackets specify the boundaries of the class declaration
  - Method names are NOT capitalised, and always immediately followed by brackets
    - there may be parameters between the brackets (but not in this example)
  - More curly brackets indicate the start and end of each method
  - Make sure your curly brackets are matched!

```
public class Ball
{
    public double diameter;
    public String colour;

    public void bounce()
    {
        System.out.println("Boing");
    }

    public void roll()
    {
        System.out.println("Whee");
    }
}
```

More on our balls

- On its own, our Ball class does nothing
- Think of a **class** as a general "description" of objects of that type
- Our class is a description of ALL balls, not any specific individual ball
- We have to create **instances of our class** or **objects** (these terms are interchangeable)
- When we do, the class is used as the "blueprint" for these new objects
- **Instances of a class (objects) are unique and individual entities**
  - **There will be one single class definition**
  - **There may be any objects of that class**

```
public class Ball
{
    public double diameter;
    public String colour;

    public void bounce()
    {
        System.out.println("Boing");
    }

    public void roll()
    {
        System.out.println("Whee");
    }
}
```

Class naming rules

- Class names, same as variable names…
  - Must begin with a letter, the dollar sign $, or the underscore character _
  - May then contain letters, digits, dollar signs, or underscores
  - May NOT contain any other characters – including spaces
  - May not be the same as a Java keyword

Class naming conventions

- Class names should always start with an upper case (large) letter
  - Person, Fruit, UniversityStudent 
  - car, house, whatIsAClassAnyway 
- Class names should try to describe the actual real world thing they represent, or describe their purpose
  - Exam, Customer, OfficeBuilding 
  - MyClass, X, Xabc1235foobar 
- If the name you choose consists of only one word, start with a capital letter, then revert to lowercase letters. If it consists of more than one word, capitalise the first letter of each subsequent word

The role of a main method

- Complex Java applications will consist of many classes
- Consider a University registration system
  - Classes in the system might include
    - Student
    - Course
    - Module
    - Tutor
    - Exam
    - Classroom
    - Coursework
    - Qualification
- Which class should the application start from when it is run? How do we know?

The role of a main method

- A main method is a special method that provides a starting point for an application
- The code in the main method can create instances of our classes (objects), which in turn might create other objects (and again, and again, and again…)
- The relationships between classes in OO programming is crucial, and is where its power lies

Making use of our balls - we need a main method somewhere

- The ball class

```
public class Ball
{
  public double diameter;
  public String colour;

  public void bounce()
  {
    System.out.println("Boing");
  }

  public void roll()
  {
    System.out.println("Whee");
  }
}
```

- A "main" class

```
public class Main
{
  public static void main (String[] args)
  {
    Ball tennisBall = new Ball();
    tennisBall.diameter = 6.5;
    tennisBall.colour = "green";

    Ball cricketBall = new Ball();
    cricketBall.diameter = 9;
    cricketBall.colour = "red";

    System.out.println("A cricket ball is");
    System.out.println(cricketBall.colour);
    tennisBall.roll();
  }
}
```

Dissecting the main method

- The ball class

```
public class Ball
{
  public double diameter;
  public String colour;

  public void bounce()
  {
    System.out.println("Boing");
  }

  public void roll()
  {
    System.out.println("Whee");
  }
}
```

Main Method

- A "main" class

```
public class Main
{
  public static void main (String[] args)
  {
    Ball tennisBall = new Ball();
    tennisBall.diameter = 6.5;
    tennisBall.colour = "green";

    Ball cricketBall = new Ball();
    cricketBall.diameter = 9;
    cricketBall.colour = "red";

    System.out.println("A cricket ball is");
    System.out.println(cricketBall.colour);
    tennisBall.roll();
  }
}
```

Dissecting the main method

- The ball class

```
public class Ball
{
  public double diameter;
  public String colour;
}
```

- A "main" class

```
public class Main
{
  public static void main (String[] args)
  {
    Ball tennisBall = new Ball();
    tennisBall.diameter = 6.5;
    tennisBall.colour = "green";

    Ball cricketBall = new Ball();
    cricketBall.diameter = 9;
    cricketBall.colour = "red";

    System.out.println("A cricket ball is");
    System.out.println(cricketBall.colour);
    tennisBall.roll();
  }
}
```

Variable declaration

Dissecting the main method

- The ball class

```
public class Ball
{
  public double diameter;
  public String colour;

  public void bounce()
  {
    System.out.println("Boing");
  }

  public void roll()
  {
    System.out.println("Whee");
  }
}
```

Object creation

- A "main" class

```
public class Main
{
  public static void main (String[] args)
  {
    Ball tennisBall = new Ball();
    tennisBall.diameter = 6.5;
    tennisBall.colour = "green";

    Ball cricketBall = new Ball();
    cricketBall.diameter = 9;
    cricketBall.colour = "red";

    System.out.println("A cricket ball is");
    System.out.println(cricketBall.colour);
    tennisBall.roll();
  }
}
```

Dissecting the main method

- The ball class

```
public class Ball
{
  public double diameter;
  public String colour;
  public void roll()
  {
    System.out.println("The ball is rolling");
  }
}
```

Assigning a new (blank) object to a variable

- A "main" class

```
public class Main
{
  public static void main (String[] args)
  {
    Ball tennisBall = new Ball();
    tennisBall.diameter = 6.5;
    tennisBall.colour = "green";

    Ball cricketBall = new Ball();
    cricketBall.diameter = 9;
    cricketBall.colour = "red";

    System.out.println("A cricket ball is");
    System.out.println(cricketBall.colour);
    tennisBall.roll();
  }
}
```

Dissecting the main method

- The ball class

```
public class Ball
{
   public double diameter;
   public String colour;

   pu
   {
      S                    ig");
   }

   pub
   {
      S                    e");
   }
}
```

Setting an attribute

- A "main" class

```
public class Main
{
   public static void main (String[] args)
   {
      Ball tennisBall = new Ball();
      tennisBall.diameter = 6.5;
      tennisBall.colour = "green";

      Ball cricketBall = new Ball();
      cricketBall.diameter = 9;
      cricketBall.colour = "red";

      System.out.println("A cricket ball is");
      System.out.println(cricketBall.colour);
      tennisBall.roll();
   }
}
```

Dissecting the main method

- The ball class

```
public class Ball
{
                                                    ing");
}
                                                    ee");
}
```

- A "main" class

```
public class Mainv
{
  public static void main (String[] args)
  {
    Ball tennisBall = new Ball();
    tennisBall.diameter = 6.5;
    tennisBall.colour = "green";

    Ball cricketBall = new Ball();
    cricketBall.diameter = 9;
    cricketBall.colour = "red";

    System.out.println("A cricket ball is");
    System.out.println(cricketBall.colour);
    tennisBall.roll();
  }
}
```

Dissecting the main method

- The ball class

```
public class Ball
{
  public double diameter;
  public String colour;

  public void roll()
  {
    System.out.println("Whee");
  }
}
```

- A "main" class

```
public class Main
{
  public static void main (String[] args)
  {
    Ball tennisBall = new Ball();
    tennisBall.diameter = 6.5;
    tennisBall.colour = "green";

    Ball cricketBall = new Ball();
    cricketBall.diameter = 9;
    cricketBall.colour = "red";

    System.out.println("A cricket ball is");
    System.out.println(cricketBall.colour);
    tennisBall.roll();
  }
}
```

what's your colour?

Accessing an attribute

Dissecting the main method

- The ball class

```
public class Ball
{
  public double diameter;
  public String colour;

  public void bounce()
  {
    System.out.println("Boing");
  }

  public void roll()
  {
    System.out.println("Whee");
  }
}
```

- A "main" class

```
public class Main
{
  public static void main (String[] args)
  {
    Ball tennisBall = new Ball();
    tennisBall.diameter = 6.5;
    tennisBall.colour = "green";

    Ball cricketBall = new Ball();
    cricketBall.diameter = 9;
    cricketBall.colour = "red";

    System.out.println("A cricket ball is");
    System.out.println(cricketBall.colour);
    tennisBall.roll();
  }
}
```

Calling a method

Using attributes in our methods

- Often, we'll want our methods to do different things depending on a specific object's state
  - Different sized balls bounce at different heights
  - If a ball rolls by quickly, the blur you see will be a different colour depending on the ball's colour

- So, we need to be able to read the attributes within our methods

Using attributes in our methods

- roll
  - rolling will now print the message
    - *colour* blur!
    - …where *colour* is the value of the appropriate attribute
- bounce
  - bouncing will now print the message
    - we bounced Xcm high
  - …where X is the diameter times two

(you get to do this one as one of the workshop exercises ☺)

Using attributes in our methods

- The ball class

```
public class Ball
{
  public double diameter;
  public String colour;

  public void bounce()
  {
    System.out.println("Boing");
  }

  public void roll()
  {
    System.out.println(this.colour+" blur!");
33 }
}
```

Using attributes in our methods

- The ball class

```
public class Ball
{
  public double diameter;
  public String colour;

  public void bounce()
  {
    System.out.println("Boing");
  }

  public void roll()
  {
    System.out.println(this.colour +" blur!");
34 }
}
```

Accessing an attribute from within a method in our class

Making use of our balls

- A "main" class

```
public class Main
{
  public static void main (String[] args)
  {
    Ball tennisBall = new Ball();
    tennisBall.diameter = 6.5;
    tennisBall.colour = "green";

    Ball cricketBall = new Ball();
    cricketBall.diameter = 9;
    cricketBall.colour = "red";

    System.out.println("A cricket ball is");
    System.out.println(cricketBall.colour);
    tennisBall.roll();
  }
}
```

```
..from Ball:
public void roll()
{
  System.out.println(this.colour+" blur!");
}
```

What will `this.colour` be in this case?

Calling a method

Remember the grammar (again!)

- Just because these examples have mostly had methods that contain one line with a `System.out.println`, doesn't mean that's ALL methods can contain!
- Your methods can - and WILL - contain `if` statements, `for` and `while` loops, variables, calculations, and everything you've seen so far

An alternative alternative bounce method

```
public class Ball
{
  public double diameter;
  public String colour;

  public void bounce()
  {
    if (this.diameter < 50)
    {
       System.out.println("Boing");
    }
    else
    {
       System.out.println("HUUUUUGE BOING!");
    }
  }

  public void roll()
  {
    System.out.println(this.colour +" blur!");
  }
}
```

The "has a" relationship

- When we define a new class, one way of thinking of it is that we've defined a new data type*
- We can use these "data types" – i.e. our classes – as attributes in other classes
- This is the way we represent two classes with a "has a" relationship

* this isn't strictly speaking *accurate*, but can be a helpful simplification

The "has a" relationship: an example

- Consider classes to represent an *employee* and a *contract*
  - Both classes have their own attributes, e.g.
    - **Employee**
      - name
      - homeAddress
      - hoursWorked
    - **Contract**
      - hourlyRate
      - annualLeave
  - We can say that *employee* HAS A *contract*
  - So *employee* has an attribute of type *contract*

The "has a" relationship: an example

- The *employee* class

```
public class Employee
{
  public String name;
  public String homeAddress;
  public int hoursWorked;

  public Contract contract;

  // an example method...
  public void printEarnings()
  {
     System.out.println("Earnings this week are:");
     System.out.println(this.hoursWorked * this.contract.hourlyRate);
41 }
}
```

- The *contract* class

```
public class Contract
{
  public double hourlyRate;
  public int annualLeave;

  // any methods would go here…
}
```

The "has a" relationship: an example

- The *employee* class

```
public class Employee
{
  public String name;
  public String homeAddress;
  public int hoursWorked;

  public Contract contract;

  // an example method...
  public void printEarnings()
  {
     System.out.println("Earnings this week are:");
     System.out.println(this.hoursWorked * this.contract.hourlyRate);
42 }
}
```

- The *contract* class

```
public class Contract
{
  public double hourlyRate;
  public int annualLeave;

  // any methods would go here…
}
```

An attribute of type *Contract*, called *contract*

The "has a" relationship: an example

- The *employee* class

```
public class Employee
{
  public String name;
  public String homeAddress;
  public int hoursWorked;

  public Contract contract;

  // an example method...
  public void printEarnings()
  {
     System.out.println("Earnings this week are:");
     System.out.println(this.hoursWorked * this.contract.hourlyRate);
  }
}
```

- The *contract* class

```
public class Contract
```

```
Rate;
ve;

 go here…
```

The "has a" relationship: an example

- The *employee* class

```
public class Employee
{
  public String name;
  public String homeAddress;
  public int hoursWorked;

  public Contract contract;

  // an example method...
  public void printEarnings()
  {
     System.out.println("Earnings this week are:");
     System.out.println(this.hoursWorked * this.contract.hourlyRate );
44 }
}
```

- The *contract* class

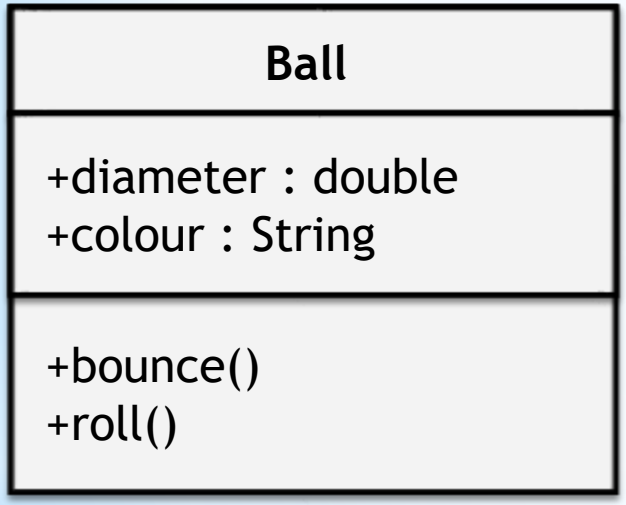
```
public class Contract
{
  public double hourlyRate;
  public int annualLeave;

  // any methods would go here…
}
```

accessing the attribute within the attribute

Introducing UML

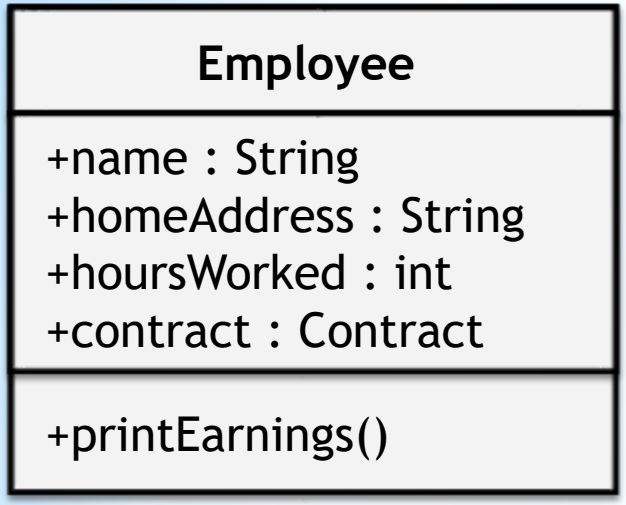
- The Unified Modeling Language
- A visual language for describing aspects of object oriented software…
  - …think diagrammatic pseudocode or flowcharts for OO
  - In a way, it's like the old blocks of *Thinking Like A Programmer* – except for OO
- One of the techniques in UML is the *class diagram*
- Class diagrams let us specify our classes and, most importantly, the *relationships* between them

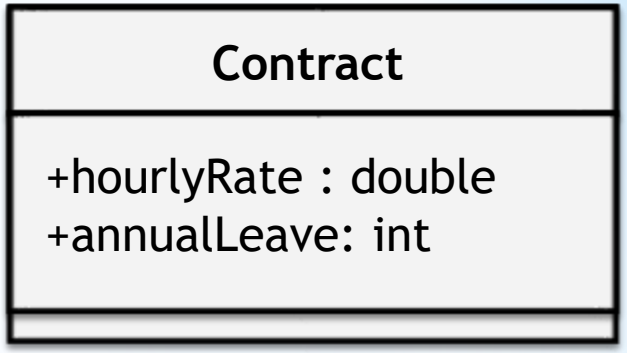
Ball in UML




```
public class Ball
{
  public double diameter;
  public String colour;

  public void bounce()
  {
    System.out.println("Boing");
  }

  public void roll()
  {
    System.out.println(this.colour +" blur!");
  }
}
```

Employee and Contract in UML

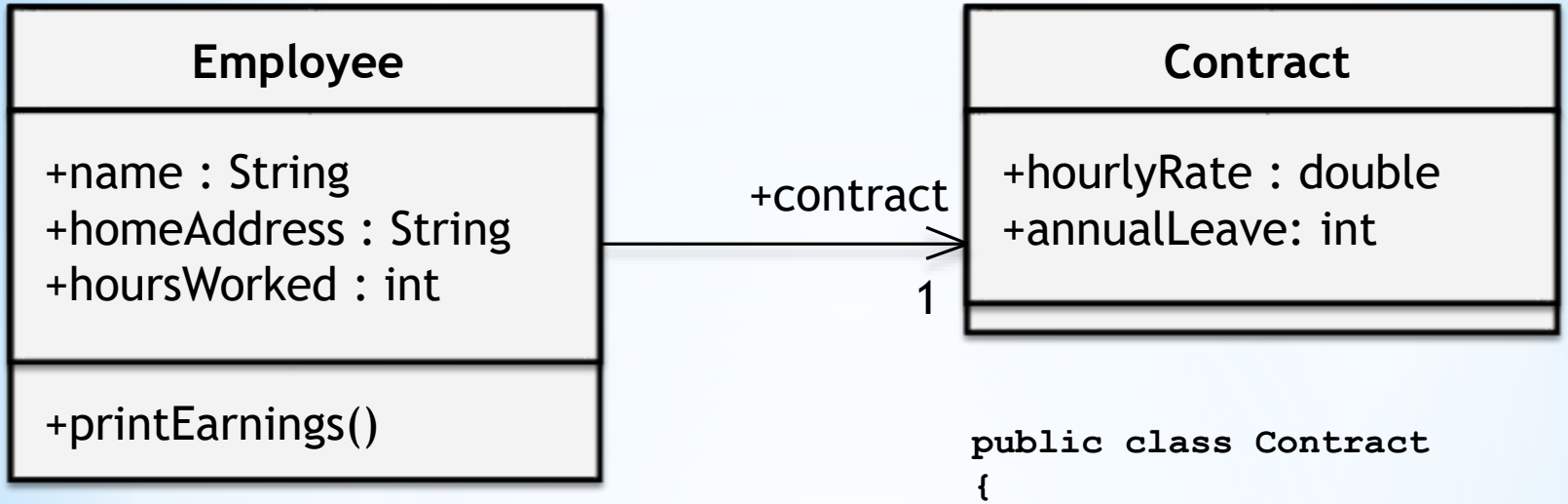





```
public class Employee
{
  public String name;
  public String homeAddress;
  public int hoursWorked;

  public Contract contract;

  // an example method...
  public void printEarnings()
  {
     System.out.println("Earnings this week are:");
     System.out.println(this.hoursWorked * this.contract.hourlyRate );
  }
}
```

```
public class Contract
{
  public double hourlyRate;
  public int annualLeave;

  // any methods would go here…
}
```

Employee and Contract in UML




```
public class Employee
{
  public String name;
  public String homeAddress;
  public int hoursWorked;

  public Contract contract;

  // an example method...
  public void printEarnings()
  {
     System.out.println("Earnings this week are:");
     System.out.println(this.hoursWorked * this.contract.hourlyRate );
  }
}
```

```
public class Contract
{
  public double hourlyRate;
  public int annualLeave;

  // any methods would go here…
}
```

Summary

- Object oriented (and, ultimately, Java) programming is really about writing collections of **classes**
  - A class is a blueprint for an **object** (or **an instance of the class**)
  - An **object** is a specific, individual, unique item of that class type
    - e.g. Ball is the class, cricket ball or tennis ball is an object of type ball

Summary

- Classes have **attributes** that store **state** and **identity**, and **methods** that describe things that objects of that class can *do*
  - A ball has a diameter (attribute)
  - Balls can bounce (method)
- Methods can behave differently for each object of that class, by using the values in their parameters

Summary

- Classes can have **relationships** with each other
- A "**has a**" relation
- is when a class has an attribute that is of another class's type
  - e.g. Employee HAS A Contract
- Use UML class diagrams to visually depict your classes and their relationships