**Lecture 2**

Object-oriented programming languages, such as Java, have the right 'building blocks' for easier modeling and hence higher quality software construction.

The world is a place where there are objects that communicate to get work done. A Java program, when it executes, also uses objects that communicate to get work done.

We discuss objects, object references, (instance) methods, events, clients, messages, parameters, results, signatures, interfaces, and sequential & concurrent execution.

We look at a trivial Java program in this new, object-oriented way.



**Objects**

- The world consists of objects
  - some tangible, e.g. you, me, this room
  - some intangible, e.g. my bank account, this lecture
- A Java program, when it executes, populates the computer memory with software objects
- The fact that we have some correspondence (even though it's not a perfect correspondence) between world objects and software objects is what makes object-oriented programming one way of reducing the conceptual gap



**Object references**

- How do we refer to an object?
- We use phrases. E.g. in the real-world, we often use names
- A **reference** is a phrase that is used to refer to an object.
- E.g. *"CS2000", "Computer Science Department", "Derek in Computer Science"*
- An object may have zero, one or more references.
- Similarly, we can write 'phrases' (expressions) in Java that refer to Java objects



**Object behaviour**

- Objects have behaviour: the things they will do and tell
- E.g. the 'tasks' a person will perform
  *deciding to re-order stock, calculating pay, revealing your height, moving an object*
- E.g. the 'functions' of a machine
  *washing whites, washing coloureds, repeating track 3 on the CD*
- E.g. less naturally,
  *my bank account object can tell me the current balance, update the balance, print a statement*
- In Java, each piece of behaviour is referred to as a **method**. (Strictly, we are talking about **instance methods** here.)



**Method invocation**

What makes an object carry out one its methods?

1. Triggered by an **event**, e.g.

| event | method invoked |
|---|---|
| *a door slams* | *I flinch* |
| *a piano falls on me* | *I swear* |

2. Object $A$, the **client**, requests the behaviour from object $B$ by sending a **message** to $B$
   - E.g. Sally in Sales sends a message to Finbarr in Finance requesting a credit check on a new customer
   - Equally an object can send a message to itself. E.g. you decide to make yourself a cup of tea (you 'send yourself a message' which invokes your tea brewing method)



**Messages**

- A message comprises at least
  1. the object to whom the message is addressed, i.e. a reference
  2. the name of the method to be performed
- E.g. Sally says

```
"Finbarr, do a credit check for me?"
   (1)               (2)
```

- E.g. I say

```
"Derek, I'll make myself a cup of tea
   (1)                   (2)
```

**Arguments/actual parameters**

- A message will also usually contain

   3. Data or objects needed for carrying out the method

- E.g. Sally actually says

```
"Finbarr, can you check the credit on
   (1)                (2)
 Customer37: is he OK for 100 quid?"
 (3) an object            (3) data
```

- The items of data are called **(actual) parameters** or **arguments**
  — there can be 0, 1, 2,... of them



**Results from methods**

- Some methods produce a result that they return to their client
  - E.g. Finbarr's credit check method returns *"OK"* or *"Not OK"* to Sally
  - E.g. my tea brewing method returns a cup of tea object

- Other methods do not return a result.



**Method signatures and object interfaces**

- If client object $A$ is to invoke one of object $B$'s methods, it must know the method **signature**
  - method name
  - how many parameters, in what order and what type of data each must be
  - the type of the result, if any

- E.g. Sally must know that Finbarr's method
  - is called "do a credit check"
  - takes in a customer and an amount of money
  - returns *"OK"* or *"Not OK"*

- The set of services an object provides to client objects is called its **interface**



**Sequential versus concurrent execution**

- Consider objects $A$ and $B$, where $A$ has sent a message to $B$, requesting that $B$ carries out some method.

- In *sequential execution*,
  - $B$ executes the requested behaviour
  - meanwhile, $A$ waits (doing nothing)
  - when $B$ halts, $A$ resumes

- In *concurrent execution* $A$ continues with other work *while* $B$ is executing the method

- We assume sequential execution for most of this course



**A Java object**

- There is a 'predefined' object which you can use in any Java program. It is referenced by the phrase `System.out` It's a *stream* along which data can flow. In the case of `System.out`, data flows to the screen.

- There are many methods in its interface, so we look at only two

◦ `public void println(String s)`
  Print the `String s`, and then finish the line
  **Parameters:**
  `s` - the `String` to be printed
  **Returns:**
  `void`

◦ `public void print(String s)`

Example of a message

```
System.out.println("Hello world");
```



**Example program**

Send messages to print some text on the screen.

```
/**
 * A program which displays some text.
 * @author Derek Bridge 666 d.bridge@cs.ucc.
 */
public class TextPrinter
{
   public static void main(String[] args)
   {







   }
}
```