

Lecture 9

Class test

- Monday Week 6, October 31st
- Test will be on aspects of procedural programming (lectures 1-8)

Seminars

- There are no seminars this (seminar) week

Note:
 Deadline for coursework 1 – Wednesday week 4 (26 October)
 Deadline for coursework 2 – Friday week 10 (2 December)
 Class test – Monday week 6 – (31 October)

CS118: Programming for Computer Scientists 1

A word on the coursework

It didn't work...

- It was a good exercise in building *prototyping* software
- The prototype did demonstrate that the specification was not a good one
- Demonstrates the difference between a good program and a good specification

The specification

- You must program to the specification
- If you think the specification is bad, go back to the customer and renegotiate the specification
- It is not your job to change the specification while you are coding

CS118: Programming for Computer Scientists 2

Paradigm shift

Procedural Programming

- Converting program specification to method-based code
- Supported by programming languages such as C, Pascal, Basic, Ada, COBOL...
- Data and operations on the data are separate

Object Oriented Programming

- Contain all the power of procedural programming
- Supported by programming languages such as Smalltalk, C++, Java...
- Data and operations are bundled together in a structure called an Object

CS118: Programming for Computer Scientists 4

Paradigm shift cont...

```
import uk.ac.warwick.dcs.util.io.IO;
public class Test
{
    public static double power(double x, double y)
    { ... }

    public static void main(String[] args)
    {
        double result;
        double input1 = IO.readdouble("Enter input 1 ");
        double input2 = IO.readdouble("Enter input 2 ");

        result = power(input1, input2);
        System.out.print("Input 1 to the power of input 2 is " + result);
    }

    // The power method definition could also have gone here
}
```

CS118: Programming for Computer Scientists 5

Paradigm shift cont...

This program can be broken down into the data

result, input1, input2, x, y

And the operations on that data

IO.readdouble, power, System.out.print

The operations modify the data (the program state)

However, operations are usually data-specific, i.e. they are only appropriate for certain data items

Yet the logical structure of the program does not reflect this

CS118: Programming for Computer Scientists 11

A simple example: a Circle

Properties (data)

- Has a radius (for example)

Behaviours (operations)

- We can calculate the area, the circumference, the diameter etc.

The properties are the data used to represent the *state* of the Circle; the behaviours are methods than can *operate* on this state

We would like to associate these data and operations somehow, and group them in a Circle thing

CS118: Programming for Computer Scientists 14

A Circle class

A **property** that characterises a circle is it's radius

A **behaviour** associated with a circle is the computation of its area

```

class Circle // a circle class
{
    double radius = 1.0; // property
    public double findArea() // behaviour
    {
        return radius * radius * 3.14159;
    }
}

```

CS118: Programming for Computer Scientists 15

A Circle class

A **property** that characterises a circle is it's radius

A **behaviour** associated with a circle is the computation of its area

```

class Circle // a circle class
{
    double radius = 1.0; // property
    public double findArea() // behaviour
    {
        return radius * radius * 3.14159;
    }
}

```

state data

method operating on state

CS118: Programming for Computer Scientists 17

Alternative thinking

This alternative way of programming can be thought of in a number of different ways. For example,

- As a way of organising your code so that the data, and the operations on that data, are bundled together
- As an extended type system. These bundles (things...) are simply more complex types (than an int or a double etc)

CS118: Programming for Computer Scientists 18

Object-oriented programming

These code bundles (like *Circle* above) allow us to create things called *Objects*

Characteristics of an object

- Properties (defined by data)
- Behaviours (defined by methods)

An object-oriented program can be viewed as a collection of cooperating objects

Examples of possible objects

- A circle
- A pack of cards
- A bank account

CS118: Programming for Computer Scientists 19

Defining objects

The code for creating an object is called a *Class* (e.g. the *Circle* class that we have just seen)

These classes are templates (or plans, or directions, or blueprints) for creating objects of this type

- by themselves they can not be run

These are different from the class which contains the main method

- which can be run
- and is referred to as the *main class*
- and is in effect *procedural*

CS118: Programming for Computer Scientists 24

From blueprint to object

So if a class by itself can not be run, how do we use this template? I.e. how do we go from a class to an object?

To produce an object we need to create an *instance* of the class:

- Declare a variable to represent the object

```

ObjectType objectName;
Circle myCircle;

```

Compare this with `int randno;`

- Create the object and assign it to the variable

```

objectName = new ClassName();
myCircle = new Circle();

```

Compare this with `randno = 0;`

CS118: Programming for Computer Scientists 27

Accessing an objects properties and behaviours

One-step declaration and creation

```
Circle myCircle = new Circle();
```

Compare this with
`int randno = 0;`

The **new** command is important

- it actually 'creates' the object
- allocates suitable memory space for the object

To access the internal components of an object the infix 'dot' is used:

```
objectName.data      myCircle.radius
objectName.method() myCircle.findArea()
```

Compare this with
`robot.getLocationX();`

```
class TestCircle // the main class
{
    public static void main(String[] args)
    {
        Circle myCircle = new Circle();
        System.out.println("Radius " + myCircle.radius +
            "area " + myCircle.findArea());
    }
}
```

```
class Circle
{
    double radius = 1.0;
    public double findArea()
    {
        return radius * radius * 3.14159;
    }
}
```

```
class TestCircle // the main class
{
    public static void main(String[] args)
    {
        Circle myCircle = new Circle();
        System.out.println("Radius " + myCircle.radius +
            "area " + myCircle.findArea());
    }
}

class Circle
{
    double radius = 1.0;
    public double findArea()
    {
        return radius * radius * 3.14159;
    }
}
```

A note on modifiers: main is a static method, findArea is not. Static associates the method with the class rather than with the object (i.e. it is more procedural in nature)

Constructors

Not all circle objects are going to have a radius of 1.0

How do we build different size circles?

Add to the class a **constructor** method

- used to build an object with particular properties (data)
- has the same name as the class

```
Circle (double r) // a constructor method
{
    radius = r; // notice it has no return type
} // and it is public by default
```

```
Circle myCircle = new Circle(5.0);
```

Constructor overloading

Can have a number of constructors in one class

```
Circle (double r) // sets the radius to the parameter value
{
    radius = r; // Circle myCircle = new Circle(5.0);
}
```

```
Circle () // sets the radius to 1.0
{
    radius = 1.0; // Circle myCircle = new Circle();
}
```

If you define a new class you should provide a constructor. If you do not, a Java default is used (data is not necessarily initialised).

Objects as arguments

Just as primitive types can be passed to methods (via parameters), so can objects

```
class TestPassingObject
{
    public static void main(String[] args)
    {
        Circle myCircle = new Circle (5.0);
        printCircle(myCircle);
    }

    public static void printCircle(Circle c)
    {
        System.out.println("Radius " + c.getRadius() +
            "area " + c.findArea());
    }
}
```

Let's recap

Sometimes we want better ways of organising or code

- Want to keep data and operations on that data together

We can do this using *classes*

- These are different from the main class

These additional classes are templates

- They provide us with a plan or a blueprint for creating and operating on some thing
- This thing is an *object*, a realisation of the plan

Objects are created (from classes) using new, e.g.

```
Circle = new Circle (5.0);
```

Let's recap

In order to control the creation of an object we define a *constructor* inside the class (determines what happens when an object of this type is built)

A class definition will also contain some properties (e.g. radius) and behaviours (e.g. findArea)

To access these elements of an object we use the *dot*, e.g.

- `myCircle.radius`
- `myCircle.findArea()`

Objects can be thought of as extensions to our type system

Before we were dealing with int and double, now we also have Circles, PacksOfCards, BankAccounts etc.