Lecture 1

Introduction to Object Oriented Programming

Methods

1

Subject Introduction

- This subject follows on from Introduction to Programming. More of the same!
- We will look at Object Oriented Programming, which is a different approach to what we have seen before.
- Everything we learned in semester 1 will be used in this subject.

2

Subject Structure

- More of the same!
- Weekly lecture, practicals, mini-tests, weekly file submissions.
- All classes delivered via Blackboard Collaborate.
- Attendance and engagement is absolutely crucial.

3

Path to success

- Attend all online sessions.
- Ask questions – use the chat function or turn on your mic to ask a question.
- Don’t be afraid to ask a question. Chances are that others will have the same issue.
- Don’t be stuck on a piece of code for more than 10 minutes. Ask for help.

4

Path to success

- Email your lecturer if you have a question of a personal / sensitive nature.
- Collaborate with your classmates.
- Don’t collaborate during assessments!
- Thursday evening – finish off all practical questions, revise notes in advance of your mini test.

5

Methods

- In semester 1 we looked at sequence, selection and iteration as forms of program control.
- Methods are another form of program control.
- Methods are named blocks of code that perform a particular action.

6

Opening Problem

To show why methods are useful we will look at the following problem:

- Find the sum of integers from 1 to 10, from 20 to 30, and from 35 to 45, respectively.

7

Problem – there is a lot of repetition

```
int sum = 0;
for (int i = 1; i <= 10; i++){
      sum = sum + i; }
System.out.println("Sum from 1 to 10 is " + sum);

sum = 0;
for (int i = 20; i <= 30; i++){
       sum = sum + i; }
System.out.println("Sum from 20 to 30 is " + sum);

sum = 0;
for (int i = 35; i <= 45; i++){
       sum = sum + i; }
System.out.println("Sum from 35 to 45 is " + sum);
```

8

Solution – use a method

```
import java.util.Scanner;
   public class SumNumbers
   {
      public static void main(String[] args)
      {
         calcSumMToN (1, 10); //call a method
         calcSumMToN (20, 30);
         calcSumMToN (35, 45);
      } //end main method

      //method
      public static void calcSumMToN(int num1, int num2)
      {
         int answer = 0;
         for (int i = num1; i <= num2; i++)
         {
            answer = answer + i;
         }
         System.out.println("Sum from " + num1 + " to "
                  + num2 + " is "  +answer);
      } //end method
   } //end class
```

9

We already use methods

- The `main()` method is used in every program

  `public static void main(String[] args)`

- *`nextInt()`* to read an *`int`* from the keyboard is a method
- *`nextDouble()`* to read a *`double`* from the keyboard is a method

- Now we will now learn how to write our own methods

10

Advantages of Using Methods

- Enables code to be reused - cutting down on code duplication
- Enables programmer to break large complex problems into smaller more manageable problems
- Facilitates debugging by localizing faulty code
- Makes code more readable
- Write a method once and reuse it anywhere.
- *Information hiding*: Hide the implementation from the user

11

Methods

- ***A Method is a named block of code that perform a specific task***
  - Each method has a name
  - Each method can be coded to perform specific tasks
  - A method can perform tasks without interference or interaction with other parts of the program
  - A method can return a value
  - A method may obtain a value from the code that calls it (it is passed as an argument)

12

Method Definition

General form of method

```
accessSpecifier returnType methodName(parameter_list)
{
  statements;
}
```

For example:

```
public static void printSmallest(int num1, int num2)
{
    if (num1 < num2)
          System.out.print("Smallest is: " + num1);
    else
          System.out.print("Smallest is: " + num2);
}
```

Methods

- The **method body** is enclosed in braces and follows immediately after the **method header (method signature)**
- When a method is *called*, execution begins at the **method body** and *terminates* when
  - a **return** statement is encountered
- **or**
  - when execution reaches the **closing brace**

14

Calling or *invoking* methods

- The statement(s) in a method are executed only when the method is ***called*** or ***invoked*** by another part of the program.

```
public static void main(String[] args)
{
  .  .  .
  printSmallest(number1, number2);
}
```

15

Calling or *invoking* methods

- When the method is ***called*** the program may send information in the form of ***arguments***.
  - In this example the arguments are *number1* and *number2*

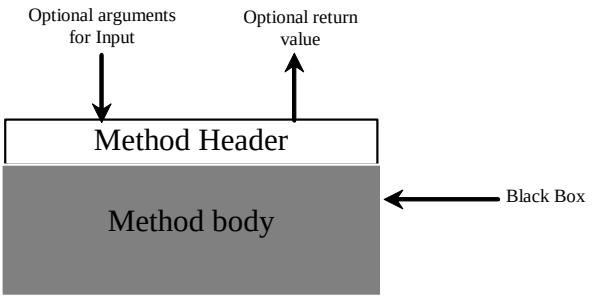
```
printSmallest(number1, number2);
```

- When the method statements have finished, execution passes back to the location where the method was called.
  - In this example, this is the *main* method

16

Method Abstraction

You can think of the method body as a black box that contains the detailed implementation for the method.



17

You try it:

```
public static void printNlines (String message, int n)
{
  for (int i = 0; i < n; i++)
      System.out.println(message);
}
```

Suppose you invoke (or call) the method using

```
printNlines("Welcome to Java", 5);
```

What is the output?

18

Answer

```
public static void printNlines (String message, int n)
{
   for (int i = 0; i < n; i++)
       System.out.println(message);
}
```

Suppose you invoke (or call) the method using

```
printNlines("Welcome to Java", 5);
```

What is the output?

```
Welcome to Java
Welcome to Java
Welcome to Java
Welcome to Java
Welcome to Java
```

19

You try it:

```
public static void printNlines (String message, int n)
{
   for (int i = 0; i < n; i++)
       System.out.println(message);
}
```

Suppose you invoke (or call) the method using

```
printNlines("Spring is coming!", 15);
```

What is the output?

20

Answer

```
printNlines("Spring is coming!", 15);
```

```
Spring is coming!
Spring is coming!
Spring is coming!
Spring is coming!
Spring is coming!
Spring is coming!
Spring is coming!
Spring is coming!
Spring is coming!
Spring is coming!
Spring is coming!
Spring is coming!
Spring is coming!
Spring is coming!
Spring is coming!
```

21

You try it:

```
public static void printNlines (String message, int n)
{
  for (int i = 0; i < n; i++)
      System.out.println(message);
}
```

Can you invoke the method using

```
printNlines(15, "Spring is coming!");
```

22

Answer

```
public static void printNlines (String message, int n)
{
   for (int i = 0; i < n; i++)
       System.out.println(message);
}
```

Can you invoke the method using

```
printNlines(15, "Spring is coming!");
```

NO. The arguments must be passed in exactly the same order as they are defined in the method header. This code generates a compiler error.

23

void Method Example

This type of method does not return a value. The method performs some actions.

```
public static void printGrade(double score)
```

This method takes a `double` as a parameter and prints a Grade (A, B, C,..) depending on the value of `score`

24

void Method Example

```
public class TestVoidMethod
{
   public static void main(String[] args)
   {
      System.out.print("The grade is ");
      printGrade(78.5); //call method
      System.out.print("The grade is ");
      printGrade(59.5);
   } //end main method

   public static void printGrade(double score)
   {
      if (score >= 90.0)
         System.out.println('A');
      else if (score >= 80.0)
         System.out.println('B');
      else if (score >= 70.0)
         System.out.println('C');
      else if (score >= 60.0)
         System.out.println('D');
      else
         System.out.println('F');
   }//end printGrade method

}//end class
```

25