Using Methods

More on writing methods

Produced by:

Dr. Siobhán Drohan
Mr. Colm Dunphy
Ms Mairead Meagher
Ms Siobhan Roche

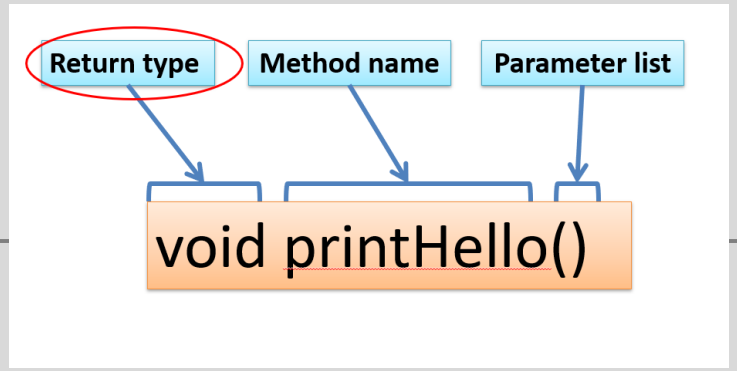
Ollscoil Teicneolaíochta an Oirdheiscirt
South East Technological University

Department of Computing and Mathematics
setu.ie

Topics list

1. Method examples – return values
   - Celcius / Farenheit **Converter**.
   - **Cubed**
   - **Product**
2. Boolean methods
3. **Overloading** methods.

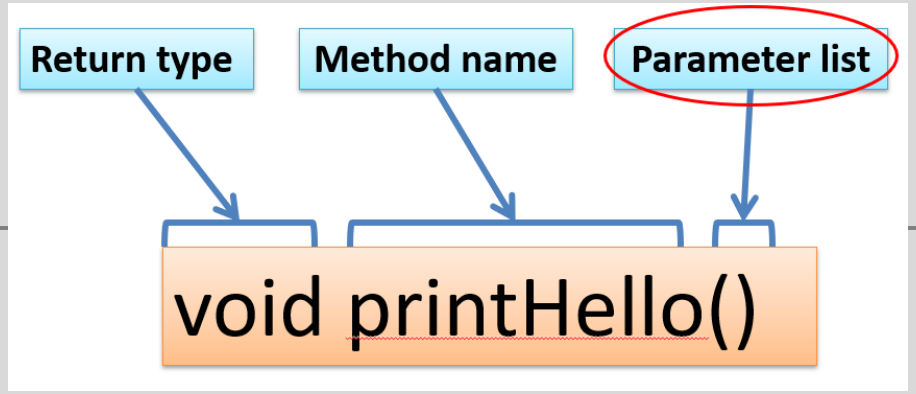
Return Type



- Methods can return information.
- The void keyword just before the method name means that **nothing is returned** from the method.
- When a data type (e.g. int) appears before the method name, this means that **something is returned** from the meth
- Within the body of the method, you use the **return** statement to **return the value**.

```
int timesTwo(int number)
{
    number = number * 2;
    return number;
}
```

Parameter list



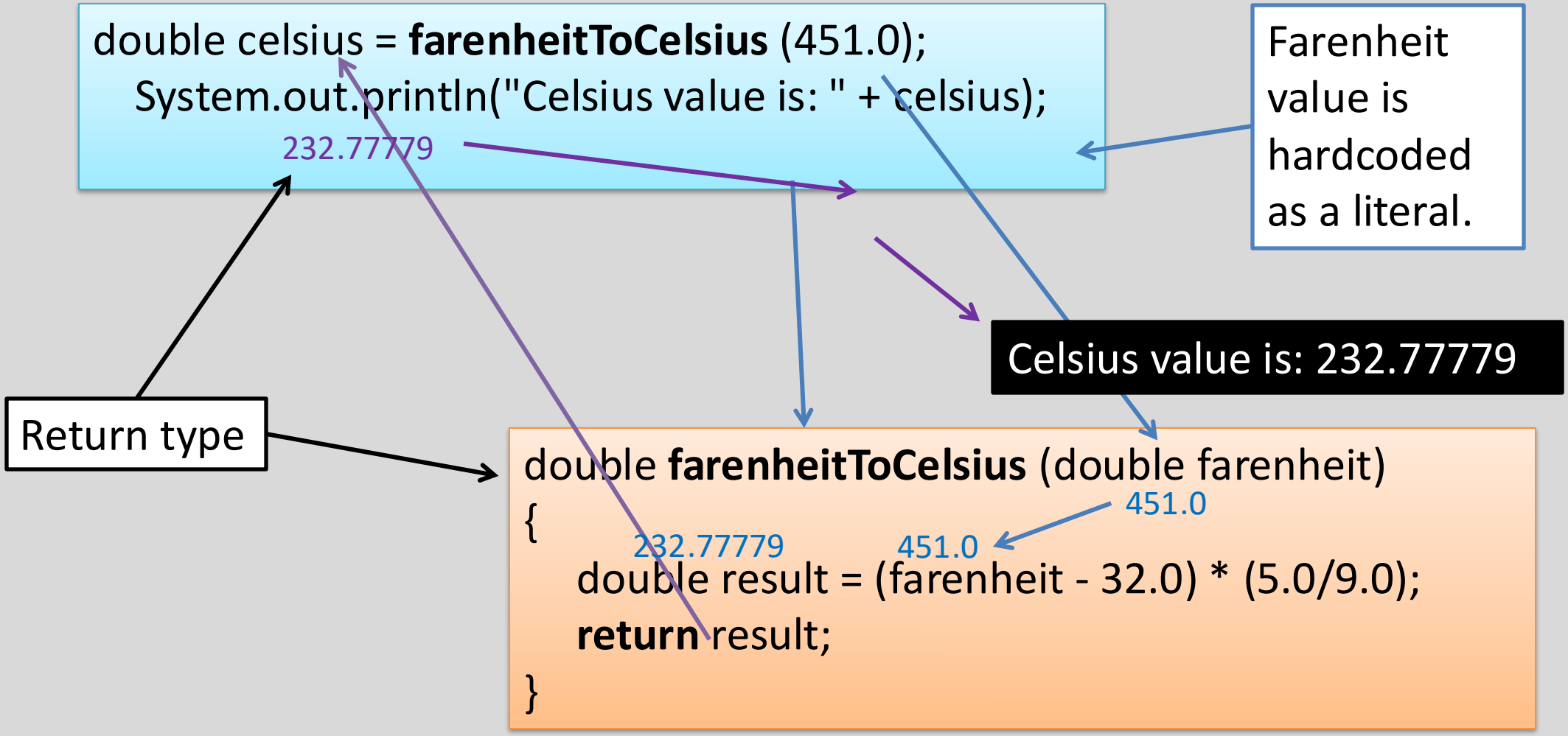
- Methods **take in data** via their **parameters**.
- Methods do not have to pass parameters
- A **parameter** is a **variable declaration** –
  - it has a **type** (e.g. int)
    and a **name** (e.g. num).
- If a method needs additional information to execute,
  we provide a parameter, so that the information can be passed into it.
- Methods can have **any number of parameters**

```
int timesTwo(int number)
{
    number = number * 2;
    return number;
}
```

Return a double

- Now we consider an example where the method **does return a value** (which could be any data type) but in this next example it returns a **double.**

Example 1 – Farenheit to Celsius



Source: Reas & Fry (2014)

Example 1 – Updated

both methods are exactly the same

```
double farenheitToCelsius (double farenheit)
{
    double result = (farenheit - 32.0) * (5.0/9.0);
    return result;
}
```

```
double farenheitToCelsius (double farenheit)
{
    return (farenheit - 32.0) * (5.0/9.0);
}
```

Source: Reas & Fry (2014)

Topics list

1. Method examples – return values
   - Celcius / Farenheit **Converter**.
   - **Cubed**
   - **Product**
2. Boolean methods
3. **Overloading** methods.

Example: **Cubed**

ClassExamples ×

Compile | Undo | Cut | Copy | Paste | Fi

```
1 public class ClassExamples {
2
3     int cubed(int num){
4         return num *5 num *5 num; 5
5     }
6
```

5

ClassExamples × | Driver ×

Compile | Undo | Cut | Copy | Paste | Find... | Close

```
1  import java.util.Scanner;
2
3  public class Driver {
4
5      public static void main(String[] args) {
6          ClassExamples ce = new ClassExamples();
7          int numToBeCubed = 5;
8          int cubedAns = ce.cubed(numToBeCubed);
9          System.out.println(numToBeCubed + " cubed is " + cubedAns);
10     }
```

125

Return type

Topics list

1. Method examples – return values
   - Celcius / Farenheit **Converter**.
   - **Cubed**
   - **Product**
2. Boolean methods
3. **Overloading** methods.

Example: Product

Driver

Compile | Undo | Cut | Copy | Paste | Find... | Close | Source Code

```
1  import java.util.Scanner;
2
3  public class Driver {
4
5      public static void main(String[] args) {
6          ClassExamples ce = new ClassExamples();
7          double a = 20.5;
8          double b = 3.5;
9          System.out.println(" The product of " + a + " and " + b + " is " + ce.product(a, b));
10
11     }
12 }
```

ClassExamples

Compile | Undo | Cut | Copy | Paste | Find... | Close

```
1  public class ClassExamples {
2      double product(double num1, double num2){
3          return num1 * num2;
4      }
5
```

Return type

```
The product of 20.5 and 3.5 is 71.75
```

Topics list

1. Method examples – return values
   - Celcius / Farenheit **Converter**.
   - **Cubed**
   - **Product**
2. Boolean methods
3. **Overloading** methods.

What is a Boolean Method?

- A Boolean method returns a boolean value — either true or false.
- They are used to check conditions and make decisions in a program.
- Examples: checking if a number is even, verifying a temperature range, or determining permissions.

Syntax – Defining a Boolean Method

```
public boolean isEven(int number) {
    if (number % 2 == 0) {
        return true;
       }
    else {
       return false;
    }
}
```

- public boolean → method returns a Boolean
- isEven → typically starts with is, has, or can

return true / false → gives back the Boolean result

```
public boolean isEven(int number) {
    if (number % 2 == 0) {
        return true;
    } else {
        return false;
    }
}
```

Long Form

```
public boolean isEven(int number
    if (number % 2 == 0) {
        return true;
    } else {
        return false;
    }
}
```

Short Form

```
public boolean isEven(int number) {
    return number % 2 == 0;
}
```

```
public boolean isEven(int number) {
    return number % 2 == 0;
}
```

Using a Boolean Method

```
public boolean isEven(int number) {
    if (number % 2 == 0) {
        return true;
       }
    else {
       return false;
    }
}
```

```
if (isEven(8)) {
    System.out.println('Number is even’);
   }
else {
    System.out.println('Number is odd');
}
```

- Output: Number is even
- Explanation: The if statement calls the Boolean method and uses its result directly.

Boolean Method with Objects

```
public boolean isAdult(int age) {
    return age >= 18;
}
```

Explanation:

The expression itself produces a true or false value.

You can return the expression directly without an if statement.

Typical Boolean Method Names

- Check if number is positive → isPositive()
- Check if user has access → hasAccess()
- Check if list is empty → isEmpty()
- Check if two values match → equals()
- Tip: Boolean methods usually start with verbs like is, has, or can.

Why Use Boolean Methods?

- ✅ Make code clearer and reusable
- ✅ Separate checking logic from actions
- ✅ Work naturally with if, while, and logical operators (&&, ||, !)

Other Examples

Check is a number between (inclusive) a lower and upper limit

```
public boolean isBetween(int numToCheck, int lower, int upper) {
    if (lower <= numToCheck && numCheck <= upper) {
        return true;
       }
    else {
       return false;
    }
}
```

```
public boolean isBetween(int numToCheck, int lower, int upper) {
    if (lower <= numToCheck && numCheck <= upper) {
        return true;
       }
    else {
       return false;
    }
}
```

Long Form

```
public boolean isBetween(int numToCheck, i
    if (lower <= numToCheck && numCheck <=
        return true;
       }
    else {
       return false;
    }
}
```

Short Form

```
public boolean isBetween(int numToCheck, int lower, int upper) {
    return(lower <= numToCheck && numCheck <= upper) ;
}
```

```
public boolean isBetween(int numToCheck, int lower, int upper) {
    return(lower <= numToCheck && numCheck <= upper) ;
}
```

Topics list

1. RECAP: Method Terminology:
2. Method examples – return values
   - Celcius / Farenheit **Converter**.
   - **Cubed**
   - **Product**
3. **Overloading** methods.

Overloaded methods

- Multiple methods can have the **same name**, once they have a **different parameter list**.
- We could write the following:
  - void addNum()
  - void addNum(int num1, int num2)
  - void addNum(int num1, int num2, int num3)
  - void addNum(int num1, int num2, int num3, int num4)

Same Name

Different Parameter List

Source: Reas & Fry (2014)

Overloaded methods

Method signature	Parameter List
void addNum()	no parameter
void addNum(int num1, int num2)	int, int
void addNum(int num1, int num2, int num3)	int, int, int
void addNum(int num1, int num2, int num3, int num4)	int, int, int, int

Same method name, but different parameter list

Source: Reas & Fry (2014)

Overloaded methods

- A program can have two or more methods with the same name, only if their parameter list is different.

- When Java is checking that a parameter list is different, it is not checking the name of the variables, it is **checking the data type** of the variables e.g.
  - this is permitted as the **data type is different**:
    - void addNum (int num1)
    - void addNum (double num1)

Data types must be different
for a method with the same name to be overloaded

Source: Reas & Fry (2014)

Overloaded methods

```
public class Driver {

    public static void main(String[] args) {
        ClassExamples ce = new ClassExamples();
        double cel = ce.farenheitToCelsius(451);
        double c = 20.5;
        double d = 3.5;
        ce.addNum(c, d);
    }
}
```

Q: Which addNum method is called and why?

```
public class ClassExamples {

void addNum(double num1, double num2){
    double result = num1 + num2;
    System.out.println("This is double addNum " +
result);
}

void addNum(int num1, int num2){
    int result = num1 + num2;
    System.out.println("This is int addNum " + result);;
}


}
```

```
This is double addNum 24.0
```

Overloaded methods

- When you call a method,
  Java **matches** the **number and type of the arguments** you passed to the method, with all the declared methods.

- When a match is found, Java invokes that method
  e.g.

  addNum (0, 4) calls void **addNum** (**int** num1, int num2)

  addNum (0.0, 4.0) calls void **addNum** (**double** num1, double num2)

Source: Reas & Fry (2014)

Key features of Method Overloading

- Multiple methods can share the same name in a class when their parameter lists are different.
- Overloading is a way to increase flexibility and improve the readability of code.
- Overloading does not depend on the return type of the method, two methods cannot be overloaded by just changing the return type.
- It does depend on:
  - The number of parameters
  - The types of parameters
  - The order of parameters

Questions?