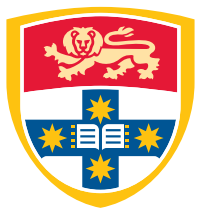
INFO1103: Introduction to Programming

School of Information Technologies, University of Sydney



Week 5: Using Objects and Methods

We will cover: Objects: purpose, creation, methods (operators). Strings, Arrays and other objects.

You should read: Savitch:

§§2.4 *reread*

§§5.3 up to page 367 with great patience

§§6.2 - 6.3 with great diligence

Lecture 9:

Methods

Methods

Building more complex programs

What are methods?

A *function* is a series of instructions that will produce an output based on a number of input. e.g. $y = sin(x)$

A *method* is a Java term to describe a function that is *associated with a class or object.*

```
1    String name = ".....";
2    int len = name.length()
```

> *A method is a separate part of a program that performs some operations, which can be invoked from somewhere else in the program.*

Methods can accept information in the form of *arguments*, and they can either return some variable or return nothing, on completion.

Why use methods?

Answer 1: it's tidy and easier to understand:

Even slightly complex programs contain nested loops, conditions, and many variables

Consider the point of view from

- Designing the new solution
- Designing new test data
- Code maintainers point of view

Who are these people?

Why use methods?

An example of using methods

```
1    if ( getKettleFilled() < 200 )
2       fillKettle();
3
4    turnKettleOn();
5
6    while ( ! isKettleBoiling() )
7       ;
8
9    turnKettleOff();
10
11   pourKettleAmount(200);
```

During design, we can identify pieces of the problem. The logic of each can be described in its own method.

Why use methods?

Answer 2: it allows for code re-use:

Don't have to reinvent the wheel each time

e.g. calculate it again, search for it again, read from input again, displaying output again

Why use methods?

Answer 3: to reduce the chance of error:

do not repeat the same code

```
1   if (model==34)  { rc = 32; pin=40; }
2   if (model==50)  { rc = 48; pin=36; }
3   ...
4   iv = (2.2 * (rc + 5)) / rc;
5   writePowerValue(pin, iv, 40, 0);
6
7   while ( ! isKettleBoiling()  )
8       ;
9
10  turnKettleOff();
11
12  pourKettleAmount(200);
13
14  if ( isKettleKeepHotOptionEnabled() ) {
15      if (model==34)  { rc = 32; pin=40; }
16      if (model==50)  { rc = 48; pin=26; }
17      ...
18      iv = (2.2 * (rc + 5)) / rc;
19      writePowerValue(pin, iv, 40, 0);
20  }
```

Why use methods?

```
1    if ( getKettleFilled() < 200 )
2       fillKettle();
3
4    turnKettleOn();
5
6    while ( ! isKettleBoiling()  )
7       ;
8
9    turnKettleOff();
10
11   pourKettleAmount(200);
12
13   if ( isKettleKeepHotOptionEnabled() )
14      turnKettleOn();
```

Any block of code required for `turnKettleOn()` is not repeated

IsNegative method

In this lecture we will give examples of static methods: functions that are associated with a *class*.
Here's a method that returns a value: when the number is less than zero it returns true, and false otherwise.

```
1  public class MethodSimple {
2     public static boolean isNegative(int n) {
3        if (n < 0) {
4           return true;
5        } else {
6          return false;
7        }
8     }
9     public static void main(String [] args) {
10       System.out.print("5 is ");
11       boolean result = isNegative(5);
12       if (result) {
13          System.out.println("negative");
14       } else {
15          System.out.println("positive");
16       }
17    }
18 }
```

Splitting code up into methods

Here's a program to print if a letter is a vowel to the console (System.out):

```
1  public class MethodSplitExample
2  {
3      public static void main(String[] args) {
4          char input = 'a';
5          if ( input == 'a' ) {
6              System.out.println('a' + " is a vowel");
7          } else if ( input == 'e' ) {
8              System.out.println('e' + " is a vowel");
9          } else if (input == 'i' ) {
10             System.out.println('i' + " is a vowel");
11         } else if (input == 'o' ) {
12             System.out.println('o' + " is a vowel");
13         } else if (input == 'u' ) {
14             System.out.println('u' + " is a vowel");
15         } else {
16             System.out.println(input + " is not a vowel");
17         }
18     }
19 }
```

Splitting code up into methods (cont.)

It seems to be doing everything at once,

1. trying to find out if it is a vowel

2. printing a specific message for each case

what if we do something else with vowels? do we just copy/paste this code and modify?

Splitting code up into methods

Identify what is the *fundamental* task...do this as a method.

```
1  public class MethodSplitExample
2  {
3      public static boolean isVowel(char ch) {
4          if ( ch == 'a' ) {
5              return true;
6          } else if ( ch == 'e' ) {
7              return true;
8          } else if (ch == 'i' ) {
9              return true;
10         } else if (ch == 'o' ) {
11             return true;
12         } else if (ch == 'u' ) {
13             return true;
14         } else {
15             return false;
16         }
17     }
```

Splitting code up into methods (cont.)

```
18      public static void main(String[] args) {
19          char input = 'd';
20          String answer = "";
21          if (! isVowel(input) )
22              answer = "not ";
23          System.out.println(input + " is " + answer + "a vowel");
24      }
25  }
```

Identify what is the *fundamental*[1] task...do this as a method.

[1] principal, essential, most important, exactly one useful thing, quintessential

Method Anatomy

There are four parts to each method:

- the method *name* (what it's called)
- the method *arguments* (the information / variables we pass it)
- the *return type* (what kind of thing is returned by the method)
- the method *body* (the actual code that does the work)

The method name and list of argument types make up the *method signature*.

Arguments and Parameters

These terms will come up often, but are often interchangeable

argument something that is *passed* to a method

parameter something that is *used* by a method

```
1  public class VariableBoxOne
2  {
3      public static void printRow(int width) {
4          for (int i = 0; i < width; ++i) {
5              System.out.print("*");
6          }
7          System.out.println();
8      }
```

```
1      public static void main(String[] args) {
2          printRow(10);
3      }
```

Returning

When you call a method you can give it information to process. e.g. a calculation $f(x)$ requires x

You also have the option of getting something back from the method – a message to say "yes it's prime" or a value that's the square root of the number given: $message = \sqrt{x}$.

Calling a Method

- Get together information I want the method to handle
- Invoke the method by using its name and supplying the information as arguments
- Do something with the returned item

Wait wait wait. . . return means print, right?

Wrong

If your code is intended to *return* something then you should NOT just print it out — this is horribly incorrect:

```
1    public void getMax(int [] nums) {
2        // ... clever code to find out the maximum
3        System.out.println(max);
4    }
```

and so is this:

```
1    public int getMax(int [] nums) {
2        // ... clever code to find out the maximum
3        System.out.println(max);
4        return max;
5    }
```

Returning from methods

Once you `return` from a method that method ceases execution.

```
1    public static int doubleMyNumber(int n) {
2        if (n < 0) {
3            System.out.println("0");
4        } else {
5            System.out.println( 2*n );
6        }
7        return 0;
8    }
```

```
1    public static int doubleMyNumber(int n) {
2        if (n < 0) {
3            return 0;
4        }
5        return (2 * n);
6    }
```

What are the differences to compare with the above methods?

Returning from methods (cont.)

What is the expected behaviour with the following?

```
1    public static int doubleMyNumber(int n) {
2        if (n < 0) {
3            return 0;
4        }
5        return ( 2 * n );
6    }
```

```
1    public static void main(String[] args) {
2        int n = Integer.parseInt(args[0]);
3        int twiceSize = doubleMyNumber(n);
4        System.out.println(twiceSize);
5    }
```

You can `return` from anywhere in a method, but be careful! once you return you can only begin the method from the very start.

Returning from methods (cont.)

Here's an example of a potential problem with return:

```
1  public static int getSumFromInput(Scanner scanner) {
2      int sum = 0;
3      int [] numbers = new int[3];
4
5      if ( !scanner.hasNext() || !scanner.hasNextInt() )
6          return sum;
7      numbers[0] = scanner.nextInt();
8
9      if ( !scanner.hasNext() || !scanner.hasNextInt() )
10         return sum;
11     numbers[1] = scanner.nextInt();
12
13     if ( !scanner.hasNext() || !scanner.hasNextInt() )
14         return sum;
15     numbers[2] = scanner.nextInt();
16
17     sum = numbers[0] + numbers[1] + numbers[2];
18     return sum;
19 }
```

Returning what?

The method return type is given at the beginning of the definition of the method. Any return statement must, *must* return something of that type.

These are not allowed:

```
1 public int foo() {
2     // ...
3     return;
4 }
```

```
1 public String [] foo() {
2     // ...
3     return "Hello, World";
4 }
```

```
1 public String foo() {
2     // ...
3     return 6;
4 }
```

```
1 public void foo() {
2     // ...
3     return "void";
4 }
```

How many things can be returned?

You can return only *one* data type.

It can be a primitive or an reference type (Object).

Don't forget an array has many elements and is just one object!

What about void?

The main method

```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         System.out.println("Hello, World!");
4     }
5 }
```

The arguments we're giving it are the `String [] args` array of `String` objects.
The `main` method returns nothing it all. To indicate this we use the keyword `void` in place of a variable type (like `int` or `String`).

For a Java program to work, it HAS to have a main method that is marked `public static void main` and MUST accept the arguments `String []`.