More about methods

Lecture 5 of TDA 540
Object-Oriented
Programming

Jesper Cockx

Fall 2018

Chalmers University of Technology — Gothenburg University

Last week: recap

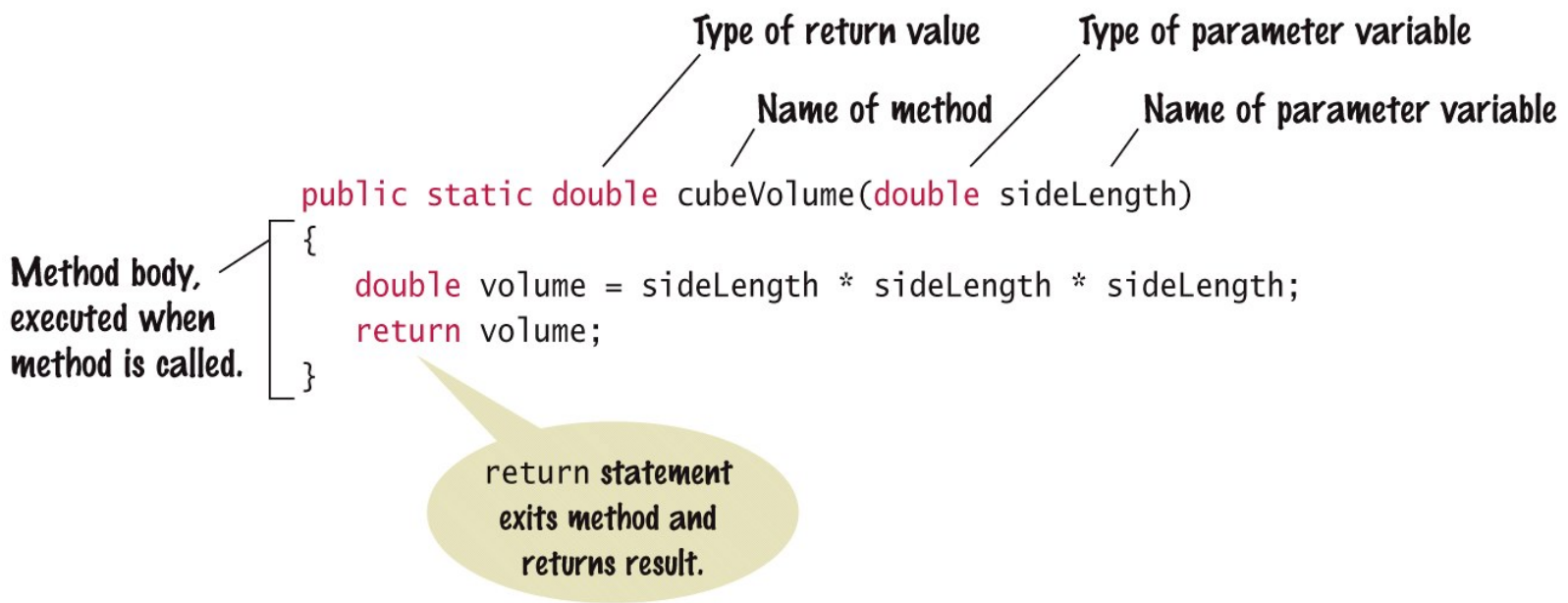
Last week's topics

- How to write your own methods
- Formal parameters vs. actual parameters
- Applying 'divide and conquer' to split a problem into smaller parts
- Notice opportunities for abstraction
- Pre- and post-conditions

Method declarations

Syntax

```
public static returnType methodName(parameterType parameterName, . . . )
{
   method body
}
```



Syntax 5.1

Formal vs actual parameters

The arguments in the method definition are the formal parameters.

```
int negate(int x) {
  return -x;
}
```

The arguments at the place where the method is used are the actual parameters.

```
int x = 5;
x = negate(x);
```

Private vs public methods

- A private method can only be used in other methods in the same class.
- A public method can be used from any class.
- (A protected method can be used from any class in the same package.)

Unless there is a good reason, most methods should be private!

Pre- and postconditions

A precondition says what should hold *before* a method is called.

A postcondition says what should hold *after* the method has completed.

```
// pre:  a and b are positive integers
// post: the result is true iff a divides b
static boolean divides(int a, int b) {
  return b % a == 0;
}
```

The DRY principle

DRY: Don’t Repeat Yourself!

Whenever you notice yourself copy-pasting a piece of code, that is a missed opportunity for introducing a new method!

Kahoot!

Methods & return statements

15 min. break

The power of abstraction

Abstraction is the most powerful tool in the programmer's toolbox:

- It allow you to think about what, not how.
- It allows you to focus on one thing at a time.
- It allows you to refine the problem step-by-step.

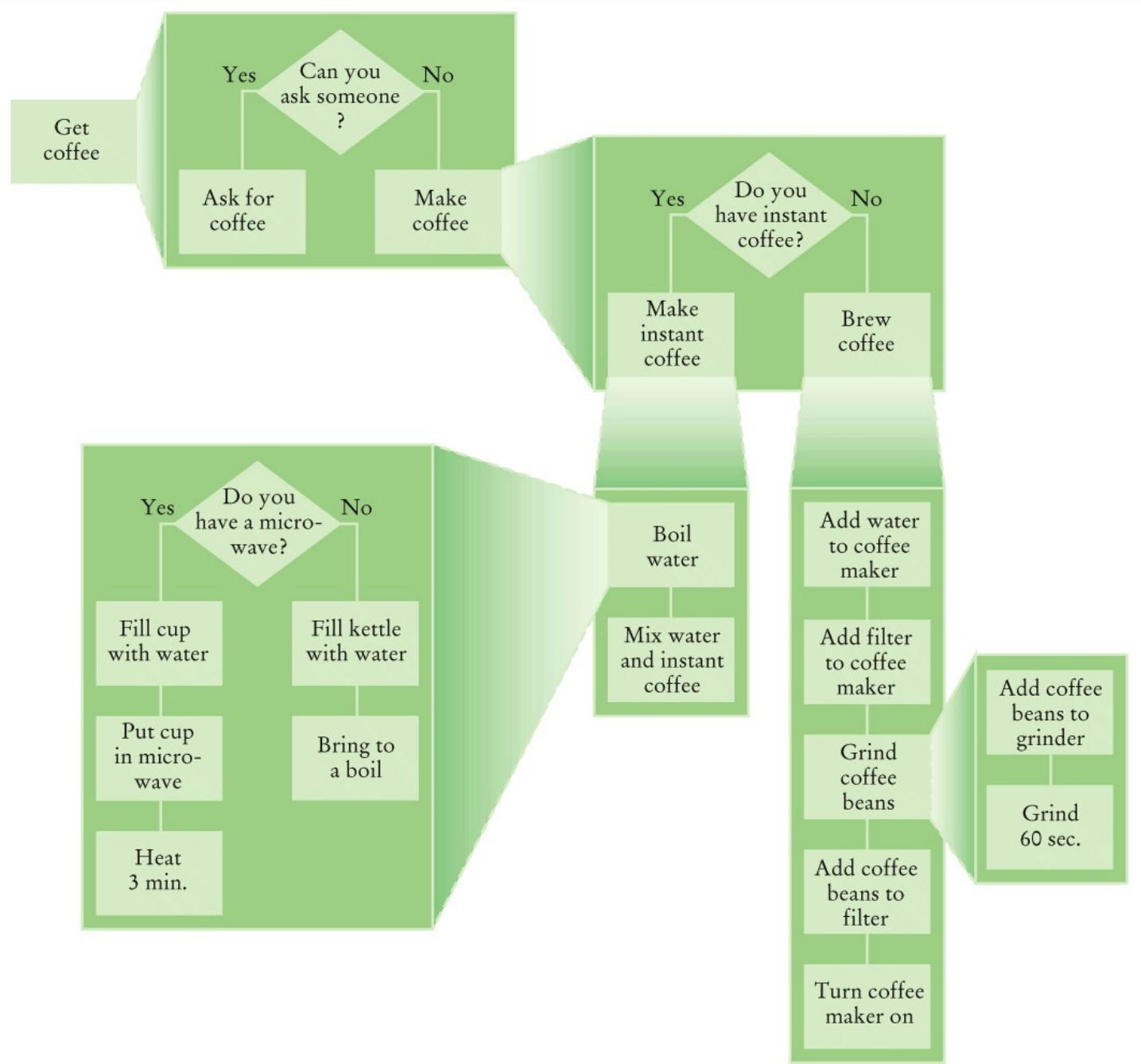


Figure 5.5

Methods as black boxes

A method = a black box:

- The user does not have to know the implementation.
- The implementation does not have to know how it is used.

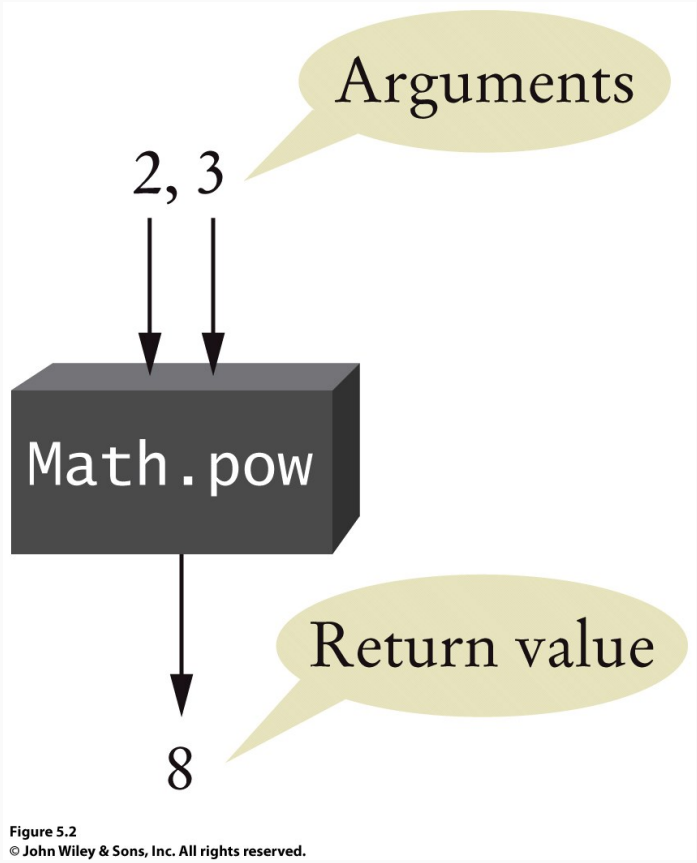


Figure 5.2

The interface of a method

Each method has an interface explaining how to use it, consisting of:

- Its name
- Whether it is `static` or an instance method
- Whether it is `private` or `public`
- Its output type
- Its arguments and their types
- Its documentation (including pre- and post-conditions)
- Possible exceptions (see later)

Try to make the most use of these!

Implementing a method: standard approach

1. Describe in words what the method should do.
    - If this is hard, apply divide and conquer!
2. Determine the inputs and outputs.
    - Return type
    - Number of arguments and their types
    - Pre- and postconditions
3. Write the method in pseudocode.
4. Implement the method.
5. Test the method.

Implementing a method: test-driven development

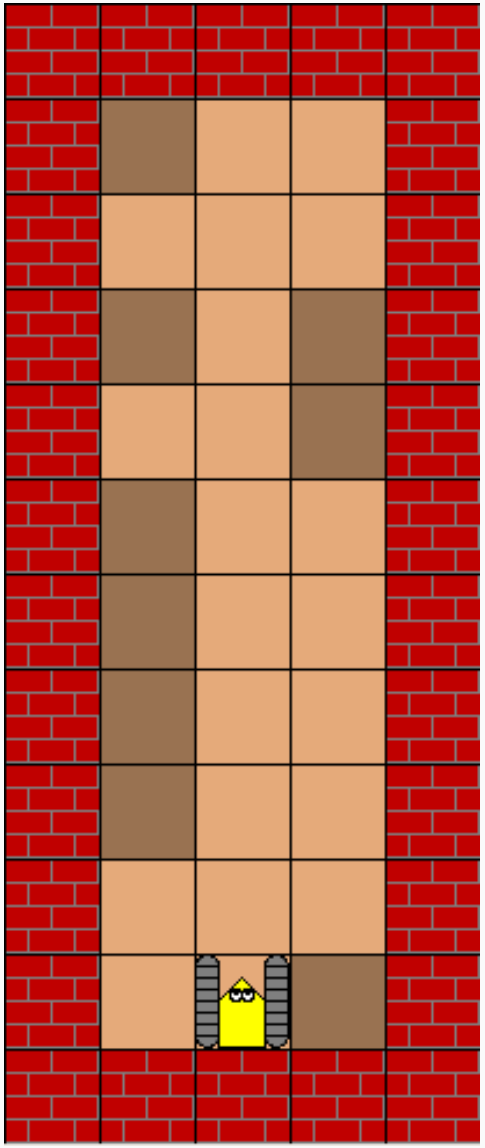
Instead of writing test at the end, you can start by writing tests.

1. Write the tests
2. If a test fails, write code until it passes
3. Refactor (clean up) the code
4. Repeat until all the tests pass

The tests become a part of the specification.

Example problem: Swapper

Assignment: program a robot to swap the cells on the left and the right of a corridor of width 3.



Think before programming

Question: What methods do we need?

One method `swapAll` that does everything in one go?

Or do we need more methods? If so, which ones?

Specification of `swapAll`

`public void swapAll()` swaps the cells to the left and the right of the corridor.

- No output value $\Rightarrow$ return type `void`
- No input values
- Precondition: the robot is at the start of a corridor of width 3.
- Postconditions:
  - The robot is at the end of the corridor
  - All rows with a dark square on one side and a dark square on the other side have been swapped.

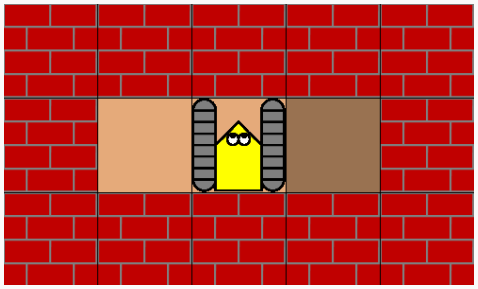
Specification of `swapAll`

`public void swapAll()` swaps the cells to the left and the right of the corridor.

- No output value $\Rightarrow$ return type `void`
- No input values
- Precondition: the robot is at the start of a corridor of width 3.
- Postconditions:
  - The robot is at the end of the corridor
  - All rows with a dark square on one side and a dark square on the other side have been swapped.

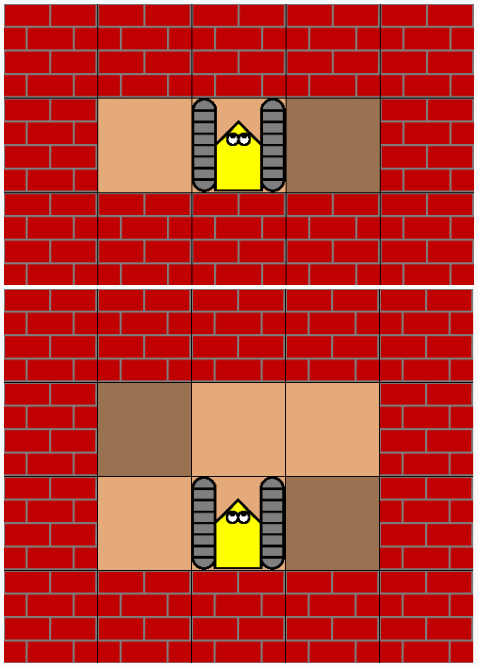
This seems hard to implement in one go...
We should apply divide and conquer!

Think of the smallest possible example



```
swapTwoCells();
```

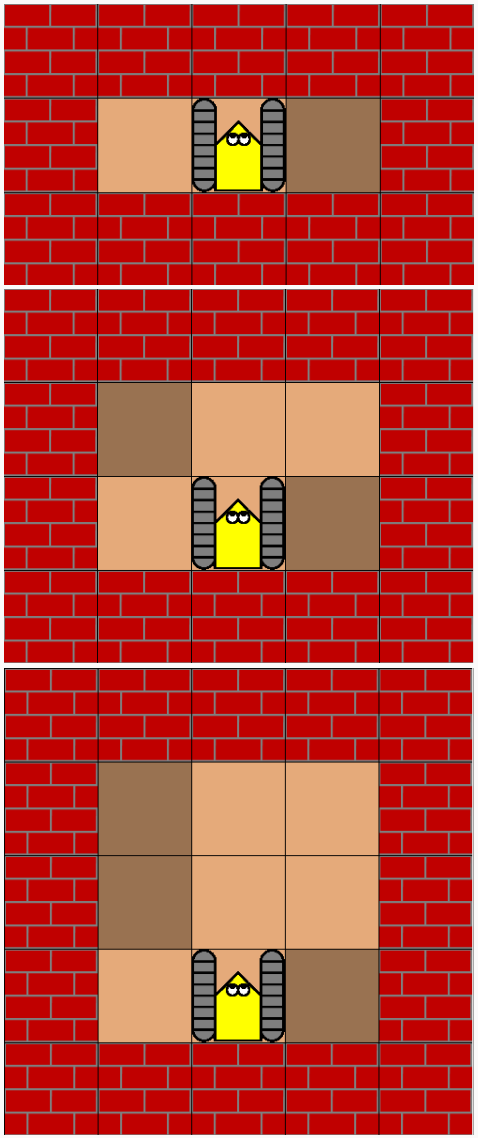
Think of the smallest possible example



```
swapTwoCells();
```

```
swapTwoCells();
robot.move();
swapTwoCells();
```

Think of the smallest possible example



```
swapTwoCells();
```

```
swapTwoCells();
robot.move();
swapTwoCells();
```

```
swapTwoCells();
robot.move();
swapTwoCells();
robot.move();
swapTwoCells();
```

Specification of `swapTwoCells`

`private void swapTwoCells()` swaps the cells to the left and the right of the robot.

- No return value $\Rightarrow$ return type `void`
- No input values
- Precondition: there are open squares to the left and the right of the robot.
- Postconditions:
  - The colors of the squares to the left and the right of the robot are swapped.
  - The robot is in the same position as where it started.

This is easier, but still quite hard...

Divide and conquer, again!

Solution: more methods!

- `areColorsDifferent` checks if colors to the left and right are different
- `changeColorOfLeft`/`changeColorOfRight` change the color of the cell to the left/right
- `changeColor` changes the color of the current cell

We also need `atEndOfCorridor` to check when the task is finished.

Specification of `areColorsDifferent`

`private boolean areColorsDifferent()` checks if colors to the left and right are different:

- Output: a `boolean` (`true` or `false`).
- Input: nothing.
- Precondition: there are open squares to the left and the right.
- Postconditions:
  - The result is `true` iff the left cell is light and the right cell is dark or vice versa.
  - The robot is in the same position as where it started.

Specification of `changeColorOfLeft`

`private void changeColorOfLeft()` changes the color of the cell to the left:

- Input and output: nothing
- Precondition: there is an open square to the left
- Postconditions:
  - The color of the cell to the left of the robot has changed from light to dark or vice versa.
  - The robot is in the same position as where it started.

Similarly for `changeColorOfRight()`.

Specification of `changeColor`

`private void changeColor()` changes the color of the cell at the current position:

- Input and output: nothing
- Precondition: none
- Postconditions:
  - The color of the cell at the current position of the robot has changed from light to dark or vice versa.
  - The robot is in the same position as where it started.

Specification of `atEndOfCorridor`

`private boolean atEndOfCorridor()` checks if the robot is at the end of the corridor:

- Output: a `boolean`
- Input: nothing
- Precondition: none
- Postcondition: the result is `true` iff the robot is at the end of the corridor.

Next step: implementation!

What's next?

Give your feedback on
`http://bit.ly/TDA540_5`.

Next lecture: **Arrays**.

To do:

- Read the book:
  - Today: chapter 5
  - Next lecture: chapter 6
- Start on the third lab:
  creating and editing music files