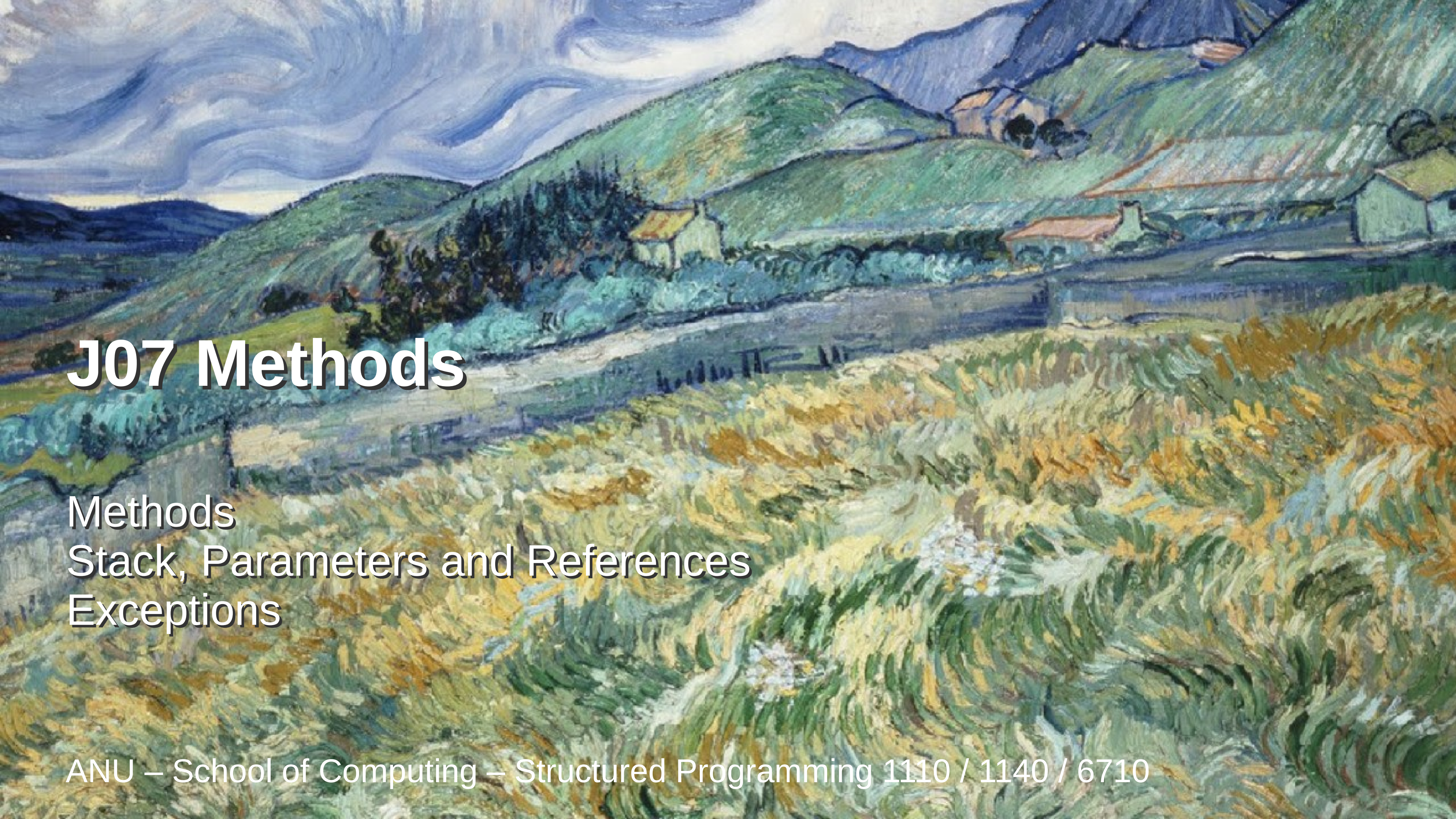


J07 Methods

Methods
Stack, Parameters and References
Exceptions

ANU – School of Computing – Structured Programming 1110 / 1140 / 6710

Methods

- A function/procedure/subroutine
  - Reusable code to perform a specific task
  - Abstraction: modularity, encapsulation
- In Java, almost all code is in a method (`main`, if not another).
- Methods may take arguments (parameters).
- Methods may return a value.

Method Declaration

Method declarations will have the following, in order:

- Any **modifiers** (`public`, `private`, `static`, etc.)
- **return type**
- **method name**
- **parameters**, in parentheses
- Any **exceptions** the method may throw
- The method **body** (code)

```
public byte[] getBytes(String charsetName)
  throws UnsupportedEncodingException {
 …
}
```

Returning a Value from a Method

The `return` statement exits the current method.

Methods `return` to caller when:

- all statements in method executed, or
- a `return` statement is reached, or
- the method throws an exception (later)

Methods declared `void` do not return a value.

All other methods must return a value of the declared type (or a subclass of the declared type, described later).

Parameters and arguments

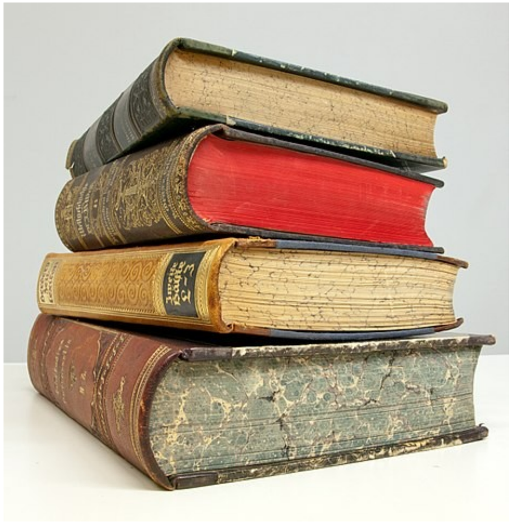
When a method is called, it must be given a list of argument expressions that match the number and types of the method’s parameters.

```
byte[] bytes = myString.getBytes(“UTF-8”);
```

Argument expressions are evaluated *before* the method is called (left-to-right) and their *values* passed as arguments.

The Call Stack: Method after Method after…

- Call Stack: a data structure that tracks method calls
  - Not directly interacted with in high-level languages like Java
  - Each call to a method pushes a *stack frame* to the stack with*:
    - **Return address:** where to continue in the **calling** method after **called** method finishes
    - The **parameters** to pass the **called** method
    - Space to store **local variables** for **called** method



By Ch. Maderthoner - Own work, CC0,https://commons.wikimedia.org/w/index.php?curid=37987842

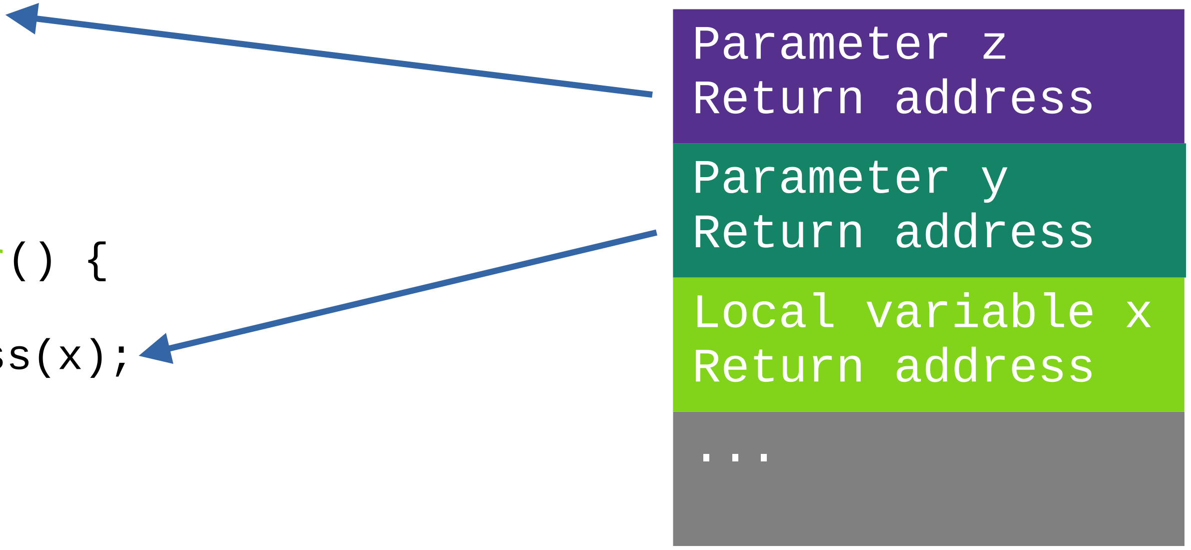
* Not specific to Java, the details depend on the language, compiler, instruction set, operating system etc...

The Call Stack

```
static int twice(int z) {
    return 2 * z;
}

static int process(int y) {
    y = twice(y);
    y = y + 1;
    return y;
}

static int number() {
    int x = 11;
    return process(x);
}
```



Parameters

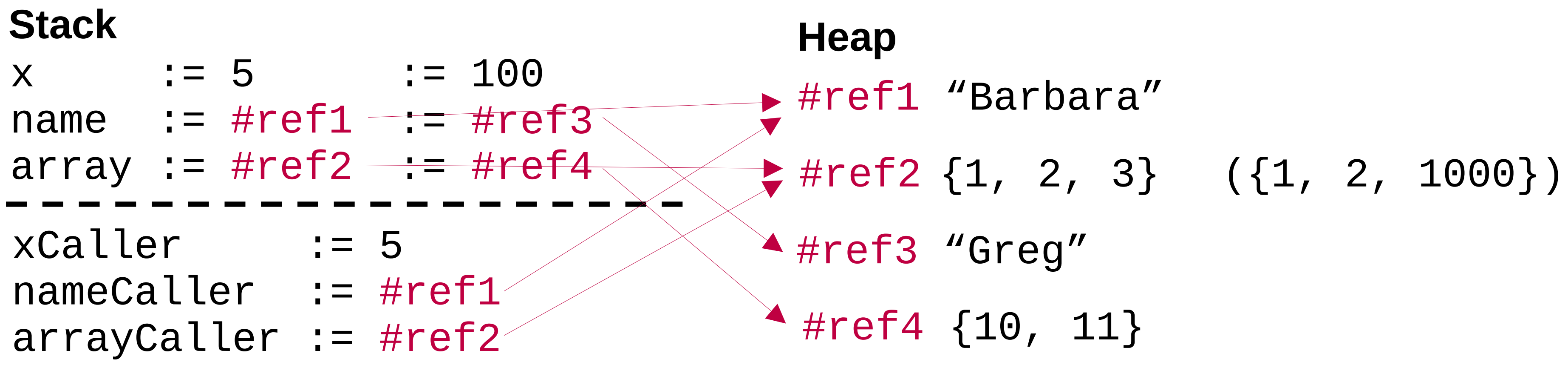
- **Primitive types** passed by value (copied into stack frame)
  - Changes to parameter are **not seen** by caller
- **References** passed by value (copied into stack frame)
  - Changes to the *reference* are **not seen** by caller
  - Changes* to *object referred* to **are seen** by caller

* Some types (e.g., String) are designed to be *immutable* – no public methods modify any class or instance fields.

Parameter Passing

```
public static void main(…) {
    int xCaller = 5;
    String nameCaller = "Barbara"
    int[] arrayCaller = new int[] {1, 2, 3};
    method(xCaller, nameCaller, arrayCaller);
}
```

```
static void method(int x,
                   String name,
                   int[] array) {
    x = 100;
    name = "Greg";
    array[2] = 1000;
    array = new int[]{10, 11};
}
```



Stack

```
x     := 5       := 100
name  := #ref1   := #ref3
array := #ref2   := #ref4
-----------------------------
xCaller     := 5
nameCaller  := #ref1
arrayCaller := #ref2
```

Heap

```
#ref1 “Barbara”
#ref2 {1, 2, 3}   ({1, 2, 1000})
#ref3 “Greg”
#ref4 {10, 11}
```

Exceptions Basics

- A method can either execute normally and return a value (passing execution back to caller), or throw an exception to signal something went wrong.
- When an exception is thrown, exception control flow kicks in: *unwinds* the call stack until either a method further down the stack “handles” the exception, or the process exits.
- We will revisit the types of exception and how to catch them later on. For now you will just likely want exceptions to crash your program so it is obvious something went wrong.

Class and Instance methods

A method declared with the `static` modifier is a **class method** (otherwise it is an **instance method**).

- Class methods
  - The method called is determined **statically** from the class of the referring variable/expression.
- Instance methods
  - The method called is determined **dynamically** from the class of the value (object) that the method is called on.
- Same with static fields.