Comparison with functional programming

Example in OCaml

```
type shape = Square of float | Circle of float
      | Rectangle of float * float;;

let perimeter = function (* perimeter : shape -> float *)
    Square side -> 4.0 *. side
  | Circle ray -> 2.0 *. Float.pi *. ray
  | Rectangle (width,height) -> 2.0 *. (width +. height);;

let area = function (* area : shape -> float *)
    Square side -> side *. side
  | Circle ray -> Float.pi *. ray *. ray
  | Rectangle (width,height) -> width *. height;;

(* example of use of shape *)
let compare s1 s2 = (* compare : shape -> shape -> int *)
   let area1=area(s1) and area2=area(s2) in
      if area1>area2 then 1 else if area1=area2 then 0 else -1;;

compare (Square 2.) (Rectangle (4.,1.))=0;;
compare (Square 2.) (Circle 1.)>0;;
compare (Circle 1.) (Square 2.)<0;;
```

Comparison with functional programming

Functional approach

- Code structured with functions (perimeter and area in the example)
- Each function uses pattern matching to deal with all kinds of data (squares, rectangles and circles in the example)

Object-oriented approach

- Code structured with classes and interfaces (`Square`, `Rectangle` and `Circle` in the example)
- Each class implements all functions (perimeter and area in the example)

Functional versus Object Programming

- functional programming: code is organized by operations (= functions)
- object programming: code is organized by data (= classes)

Interfaces and subtyping

Example with `Shape`

If an expression has static type `Shape`, then its value can be

- either a reference to an object of a class which implements `Shape` (`Square`, `Rectangle` and `Circle` in the example)
- or **`null`**

Remark

- it is not possible to create objects from interface `Shape`
- `Square`, `Rectangle` and `Circle` are subtypes of `Shape`

Interfaces and object methods

Example 1

```
public class TimerClass implements Timer {
    private int time = 60;
    ...
    public TimerClass(Timer otherTimer) {
        this.time = otherTimer.getTime(); // which method 'getTime' is called?
    }
    public int getTime() { return this.time; }
    ...
}
public class AnotherTimerClass implements Timer {
    private int minutes = 1;
    private int seconds;
    ...
    public int getTime() { return this.seconds + 60 * this.minutes; }
    ...
}
TimerClass t1 = new TimerClass();
AnotherTimerClass t2 = new AnotherTimerClass();
TimerClass t3 = new TimerClass(t1); // 'getTime' in 'TimerClass' is called
TimerClass t4 = new TimerClass(t2); // 'getTime' in 'AnotherTimerClass' is called
```

Interfaces and object methods

Example 2

```
public class ShapeComparator {
    public int compare(Shape s1, Shape s2) {
        double area1 = s1.area(); // which method 'area' is called?
        double area2 = s2.area(); // which method 'area' is called?
        return area1 > area2 ? 1 : area1 == area2 ? 0 : -1;
    }
}
ShapeComparator c = new ShapeComparator();
assert c.compare(new Square(2), new Rectangle(4, 1)) == 0;
assert c.compare(new Square(2), new Circle(1)) > 0;
assert c.compare(new Circle(1), new Square(2)) < 0;
```

Important remark

The called object method depends on the dynamic type of the target object

Reminder

Target object = the object on which the method is called = **`this`**

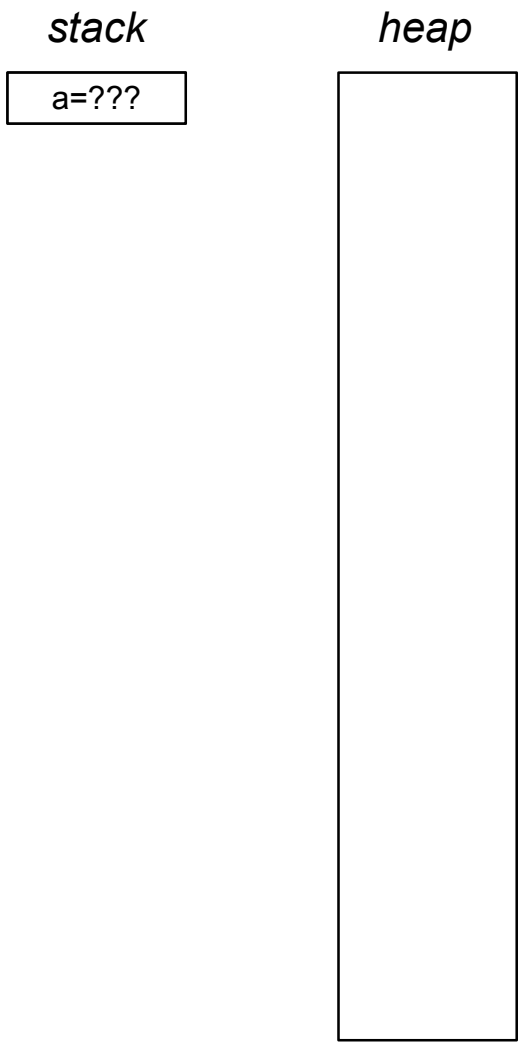
Arrays in Java

A first example

```
public class ArrayUtils {
  public static void init(int[] a) {
      // standard 'for' syntax
      for (int i = 0; i < a.length; i++)
          a[i] = i + 1;
  }
  public static int sum(int[] a) {
      int sum = 0;
      // 'enhanced for' with more compact syntax
      for (int el : a)
          sum += el;
      return sum;
  }
  public static void main(String[] args) {
      int[] a;                          // a will refer to an array of integers
      a = new int[10];                  // an array of ten integers is dynamically
           created
      assert a.length == 10;
      assert ArrayUtils.sum(a) == 0; // default value for int arrays is 0
      ArrayUtils.init(a);
      assert ArrayUtils.sum(a) == 55;
  }
}
```

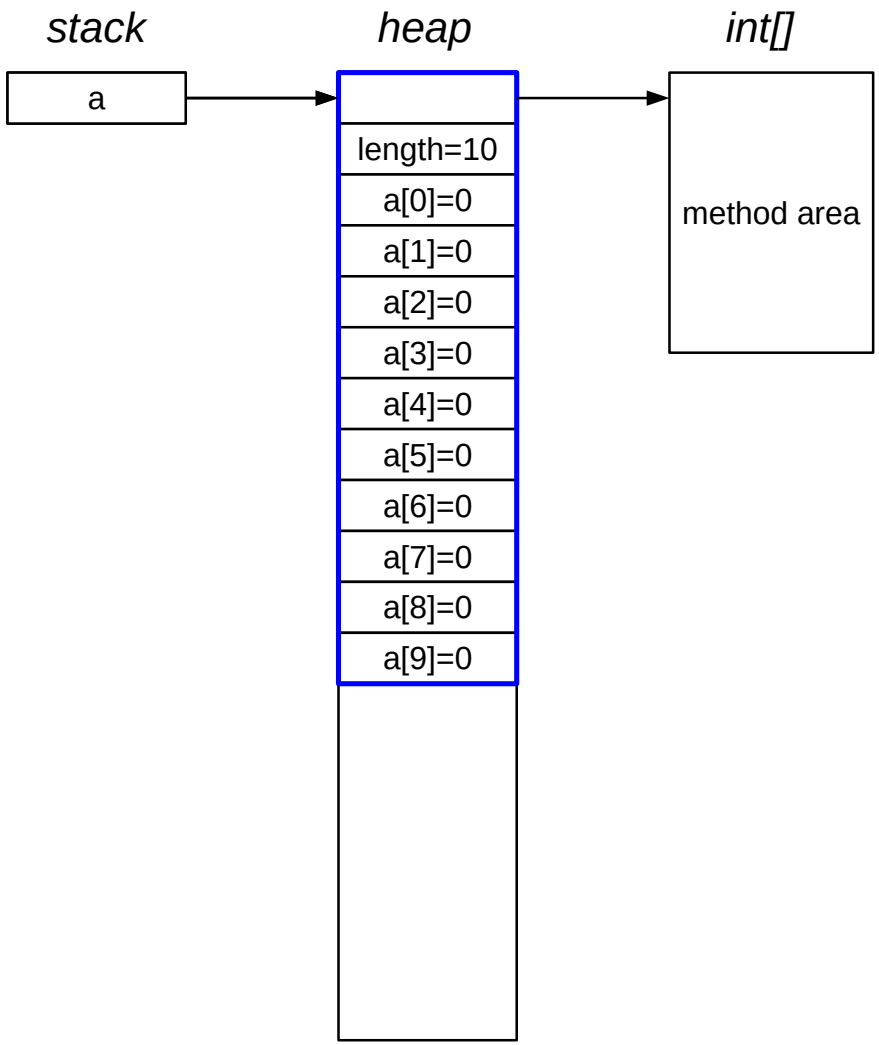
Example with memory model

```
int[] a; ⇐
a = new int[10];
ArrayUtils.init(a);
```



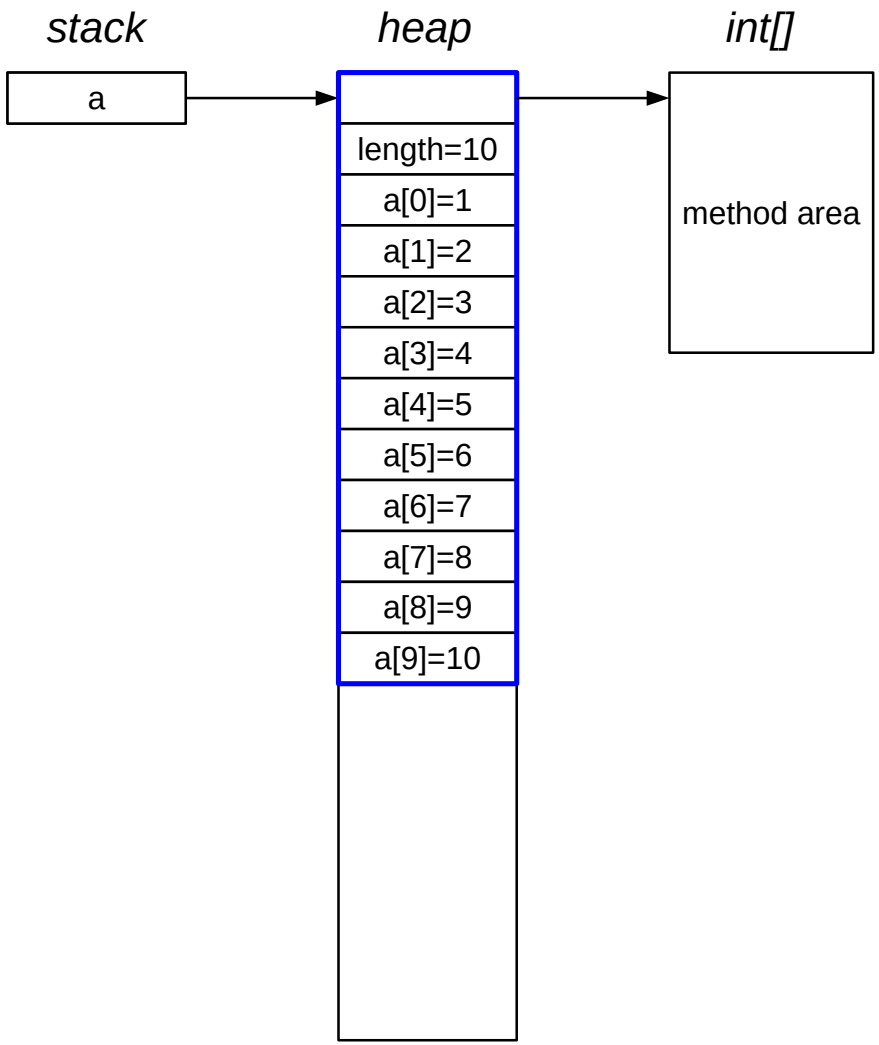
Demo

```
int[] a;
a = new int[10]; ⇐
ArrayUtils.init(a);
```



Demo

```
int[] a;
a = new int[10];
ArrayUtils.init(a); ⇐
```



Arrays in Java

Details

- arrays are references to special modifiable objects
- array components: indexed object fields with no name
- arrays can only be created dynamically
- at creation time the length (=number of components) is specified
- the length is a non negative integer and cannot change over time
- `length` is a final field of the array
- components are initialized with their default values
- components are referenced with indices from 0 to `length-1`
- `T[]` is the type of arrays with component type `T`

Arrays with object components

Example

```
String[] a = new String[3];
for(String el : a)
    assert el == null;
a[0] = "zero";
a[1] = "one";
a[2] = "two";
```

Array initialization

Demo

```
public class ArrayUtils {
    public static int sum(int[] a) {
        int sum = 0;
        // 'enhanced for' with more compact syntax
        for (int el : a)
            sum += el;
        return sum;
    }
    public static void main(String[] args) {
        // initializer at declaration time
        int[] a = {1, 2, 3};
        assert ArrayUtils.sum(a) == 6;
        // initializer at creation time
        assert ArrayUtils.sum(new int[]{1, 2, 3, 4}) == 10;
    }
}
```

Remark

Arrays are always created dynamically also with declaration initializers

Multi-dimensional arrays

Example

```
public class MultiDimensionalArray {
    public static void main(String[] args) {
        int[][] mat1 = new int[3][2]; // a matrix 3x2
        assert mat1.length == 3;
        for (int[] row : mat1) {
            assert row.length == 2;
            for (int el : row)
                assert el == 0;
        }
        int[][] mat2 = new int[3][];  // only the second size is optional
        assert mat2.length == 3;
        for (int[] row : mat2)
            assert row == null;
        // an array with variable length rows
        int[][] mat3 = { { 1, 1 }, { 1, 2, 1 }, { 1, 3, 3, 1 } };
    }
}
```

Remarks

- multi-dimensional arrays are just arrays of arrays
- dimension can be larger than 2

Main method in Java

In a nutshell

- a Java program can start execution only from a class with a main method
- a main method must always have this form:

```
public static void main(String[] args){...}
```

- command-line arguments are passed to `arg` as an array of strings

Standard output in Java

In a nutshell

- `System.out` refers to the standard output
- `System` is a predefined class
- `System.out` is a final class field of `System`
- `System.out` has type `PrintStream`
- `PrintStream` is a predefined class of the I/O Java library

Methods of `PrintStream` objects to print values

```
void println()
void println(boolean b)
void println(char c)
void println(char[] s)
void println(double d)
void println(float f)
void println(int i)
void println(long l)
void println(Object obj)
void println(String s)
```

```
void print(boolean b)
void print(char c)
void print(char[] s)
void print(double d)
void print(float f)
void print(int i)
void print(long l)
void print(Object obj)
void print(String s)
```

Basic I/O in Java via argument passing

Program demo 1

```
public class PrintArguments {
    public static void main(String[] args) {
        for (String s : args)
            System.out.println(s); // prints with new line
    }
}
```

Program demo 2

```
public class PrintArguments {
    public static void main(String[] args) {
        for (String s : args)
            System.out.print(s+" "); // prints with blank
    }
}
```

Compile and run your first Java program

Requirements

- we will do this on a terminal (or command prompt)
- you do not need an IDE
- use a text editor
- use Java SE Development Kit 19 (or earlier)

Instructions for class `PrintArguments`

- create file `PrintArguments.java` with the editor and copy the class
- compile the file with `javac PrintArguments.java`
- run with `java PrintArguments`

Remark

The name of the `.java` file and the contained public class must be the same