(remedial) Java

anastasia.bezerianos@lri.fr

Remedial Java class

- Objective:
  - Cover the fundamentals of the Java programming language.
  - Designed for students with some programming experience, but ...

- Practical info:
  contact Anastasia by email, using [RJ] in the title
  www.lri.fr/~anab/teaching/remedial-java/

- 6 sessions of 3h (theory + practice):
  - Programming basics (Mon 2/9, pm)
  - Java Language basics (Tue 3/9, am)
  - Object oriented programming (Tue 3/9, pm)
  - Inheritance (Tue 5/9 am)
  - Packages and Exceptions (Tue 5/9 am)
  - Independent work (Wed 6/9 – no class)
  - I/O (Thu 7/9 pm)
Programming basics

What is a program

- **Algorithm**: A recipe / sequence of steps to follow. "A step-by-step procedure for solving a problem or accomplishing some end, especially by a computer" (Merriam-Webster Online)

- **Program**: A sequence of instructions in a programming language that perform a task (e.g., follow an algorithm)

*(pseudo-code)*

- get from user a first number
- get from user a second number
- add the first and second number
- return to user the result
- add to the result 2 and return the new result

Each of the steps are called **instructions** of the program
What is a program

> Please give a number: 2
> Please give a second number: 3
> Result is 5
> New result is 7

> Please give a number: 7
> Please give a second number: 6
> Result is 13
> New result is 15

Same Program, two runs

get from user a first number
get from user a second number
add the first and second number
return to user the result
add to the result 2 and return the new result

What is a variable

- a variable is a storage space for keeping information that may change every time we run the program, or even during the same program (thus their name
- they are temporary and store additional information the algorithm needs to function

get from user a first number
let variable1 = first number
get from user a second number
let variable2 = second number
let variable3 = variable1 + variable2
return to user variable3
let variable3 = variable3 + 2
return to user variable3

variables are "shortcuts" to numbers we stored
What is a variable

> Please give a number:
> 2
> Please give a second number:
> 3
> Result is 5
> New result is 7

Same output as before, different structure (use of variables)

get from user a first number
let variable1 = first number
get from user a second number
let variable2 = second number
let variable3 = variable1 + variable2
return to user variable3
let variable3 = variable3 + 2
return to user variable3

What is a function / method

- It is a sub-algorithm that does a specific task, to help break down large programs into small parts
- It is a group of steps, to which we assign a name
- We can call these steps with their name

function add (variable1, variable2)
let variable3 = variable1 + variable2
return to user variable3

get from user first number and second number (e.g, 3 and 4)
let variable1= first number and variable2 = second number
add (variable1, variable2)
get from user first number and second number (now 6 and 5)
let variable1= first number and variable2 = second number
add (variable1, variable2)
What is a function / method

function add (variable1, variable2)
    let variable3 = variable1 + variable2
    return to user variable3

get from user firs number and second number *(e.g, 3 and 4)*
let variable1= first number and variable2 = second number
add (variable1, variable2)
get from user firs number and second number *(now 6 and 5)*
let variable1= first number and variable2 = second number
add (variable1, variable2)

conditionals

- When an algorithm needs to make a decision we use condition statements

```java
function add (variable1, variable2)
    variable3 = variable1 + variable2
    if (variable3 > 10)
        return to user “Too big !!!”
    else
        return to user variable3
```

get from user firs number and second number *(e.g, 3 and 4)*
let variable1= first number and variable2 = second number
add (variable1, variable2)
get from user firs number and second number *(now 6 and 5)*
let variable1= first number and variable2 = second number
add (variable1, variable2)
conditionals

```java
> Please give two numbers: 
3,4
> Result is 7

> Please give two numbers: 
6,5
> Result is “Too big !!!”
```

```java
function add (variable1, variable2) 
  variable3 = variable1 + variable2 
  if (variable3 > 10) 
    return to user “Too big !!!” 
  else 
    return to user variable3 

get from user first number and second number (e.g, 3 and 4) 
let variable1= first number and variable2 = second number 
add (variable1, variable2) 

get from user first number and second number(now 6 and 5) 
let variable1= first number and variable2 = second number 
add (variable1, variable2)
```

repetitions

- An action performed multiple times, either while a condition holds, or for a specific number of times

```java
function loop (variable iterations) 
  for iterations times 
    say Hi!

get from user (variable) iterations(e.g, 3) 
loop (iterations) 
loop (iterations + 1)
```
repetitions

function loop (variable iterations)
  for iterations times
  say Hi!

get from user (variable) iterations (e.g., 3)
loop (iterations)
loop (iterations + 1)

Please give iterations:
3

Hi!
Hi!
Hi!
Hi!
Hi!
...

value 3 stored in variable iterations
function loop is called with argument iterations
first time
second time
third time (that is equal to iterations)
function loop called again with argument iterations +1
How many Hi!’s will we see?

Java basics
Java

- Multi-platform “Pure” object-oriented language and application runtime environment
- Language syntax based on C/C++ to be familiar
  - with simplifications: no unsigned values, pointers, ...
  - developers not responsible for memory management

- Pure object-oriented language:
  - build from the ground up with OOP design in mind
  - programs are completely constructed with objects
    (as opposed to Bolt-on languages, like C++, where OO structures are an enhancement to the language, and there is a mix of procedural and OO)

Java and JVM

- Code in Java does not compile to machine code directly, but to a specific type of Bytecode
  - Bytecode executable on any architecture
  - Write once, run anywhere (WORA)

- Java bytecode runs through a Java Virtual Machine (JVM), achieving cross-platform support

- Different platforms have their implementation of the JVM (included in the Java Runtime Environments JRE). The Java Development Kit (JDK) also includes the Java JRE.
Compiling Java

Source code (.java) → javac → Bytecode (.class) → java → Java Archive (.jar)

First Java program

In a simple text editor (Notepad, TextEdit, Emacs) write the following and save in HelloWorld.java

```java
class HelloWorld {
    public static void main (String[] arguments){
        System.out.println("Hello World !");
    }
}
```

This is a class declaration, more on that later.
First Java program

In a simple text editor (Notepad, TextEdit, Emacs) write the following and save in HelloWorld.java

class HelloWorld {
    
    public static void main (String[] arguments){
        
        System.out.println("Hello World !");
    
    }

}

Main method: entry point to the program (public + static) and needs to be inside a class

System is a class, that calls out that represents the stdout (here output to console). Method println prints the text argument, and adds a newline character.
First Java program

In a simple text editor (Notepad, TextEdit, Emacs) write the following and save in HelloWorld.java

```java
class HelloWorld {
    public static void main (String[] arguments)
    {
        System.out.println("Hello World !");
    }
}
```

Let's compile and run it!

```
> javac HelloWorld.java
> java HelloWorld
```

Program structure

```java
class CLASSNAME {
    public static void main (String[] arguments)
    {
        // STATEMENTS
        // comments
    }
}
```

Main function is needed to run a program
It is always inside a class
It is always public static
Let’s move to our IDE

- Run Eclipse, and File → New → Java Project
  - Name your project: RemedialJava and click finish
- Select in the Package Explorer your project RemedialJava, and File → New → Package
  - Name your package: session1.introduction
- Finally, select your package in the Package Explorer, File → New → Java Class
  - Name your class HelloWorld
- Copy paste the code from HelloWorld example
- Project → Build and Run → Run your code (or select the HelloWorld class and Run → Run As → Java Application)

Types

- As in all programming languages, Java has some basic types, the kinds of values that can be stored and manipulated:

  boolean: Truth value (true or false).
  
  int: Integer (0, 1, -30).
  
  double: Real number (3.14, 2.0, -4.1).
  
  String: Text (“hello”, “cat”).
Variables

- **Variable**: location that stores a value of a type.
  The form is: **TYPE NAME**;
  e.g., `String cat;`

- We use `=` to make variable assignments.
  e.g., `String cat;
  cat = “Garfield”;`

- Can combine declaration and assignment
  e.g., `String cat = “Garfield”;`

Variables of different Types

- example

```java
class HelloWorld {

    public static void main (String[] arguments){

        System.out.println("Hello World !");

        String cat = "Garfield";
        int age = 3;

        System.out.println(
            "My cat is " + cat + " and he is " + age);
    }
}
```
Operators

- Symbols:
  - Assignment: =
  - Operators: +,-,*,/,%
  - Combined: +=,-=,*=,/

- Ordering:
  - 1. left to right, () increase precedence
  - 2. multiplication/division
  - 3. addition/subtractions

```java
class SomeMath {
    public static void main (String[] arguments){
        double calculate = 1.0 + 2.0 * 3.0;
        // different from (1.0 + 2.0) * 3.0
        System.out.println(calculate);
        calculate = calculate * 2.0;
        // calculate *= 2.0; would work too
        System.out.println(calculate);
    }
}
```

Division

- Division ("/") operates differently on integers and doubles

Example

```java
double a = 5.0/2.0;  // a = 2.5
int b = 4/2;         // b = 2
int c = 5/2;         // c = 2
double d = 5/2;      // d = 2.0
```
types

- Java verifies that types match:
  String five = 5; // ERROR!

- Can convert types by casting
  ```java
  int a = 2; // a = 2
  double a = 2; // a = 2.0 (Implicit)
  int a = 18.7; // ERROR
  int a = (int)18.7; // a = 18
  double a = 2/3; // a = 0.0
  double a = (double)2/3; // a = 0.6666...
  ```

String concatenation

- Basic concatenation
  ```java
  String text = "hello" + " world,"
  text = text + " times " + 6;
  // text = "hello world, times 6"
  ```

- Can also create formatted strings
  ```java
  String text = String.format("Printing a string variable %s, and an integer one %d", stringVar, intVar);
  ```

- To check equality: string1.equals(string2)
- Many methods available !!!
Exercise 1


Reminder:
- Types (boolean, int, double, String)
- Operators +,-,*,,% 
- Assignment =, +=, *=, -=, /=
- Standard output `System.out.println(“some text”);`

Methods
- A method is similar to a function
- A section of a program that is given a name, and that that performs a specific task
- It may take some input parameters and may return a value
- It can be called from elsewhere in the program to be executed
Methods

```java
public static void main(String[] arguments){
    System.out.println("hi");
}
```

```java
public static void NAME (TYPE NAME) {
    STATEMENTS;
}
```

To call a method:

```java
NAME (EXPRESSION);
```

Methods example (1)

```java
class Square {
    public static void printSquare (int x) {
        System.out.println (x * x);
    }
    public static void main(String[] arguments){
        int value = 2;
        printSquare (value);
        printSquare(value*2);
        printSquare(3);
    }
}
```
Methods example (2)

class Square {
    
    public static void printSquare (int x) {
        System.out.println (x * x);
    }

    public static void main(String[] arguments){
        printSquare ("Hello World");
        printSquare(5.5);
    }
}

What is wrong here?

Methods example (3)

class Square {
    
    public static void printSquare (double x) {
        System.out.println (x * x);
    }

    public static void main(String[] arguments){
        printSquare(5);
    }
}

What is wrong here?
Methods example (4)

class Square {
    
    public static void printSquare (double x) {
        System.out.println (x * x);
    }
    public static void printSquare (String x) {
        System.out.println (x + x);
    }

    public static void main(String[] arguments){
        printSquare(5.0);
        printSquare("Hello");
    }
}

Method overloading: a class can have two or more methods having same name, if their argument lists are different.
Method return values

public static TYPE NAME (TYPE NAME, TYPE NAME, ...) {
    STATEMENTS;
    return EXPRESSION;
}

- void means “no type”, ie nothing returned

Methods example (5)

class Square {
    public static double square (double x) {
        return x * x;
    }

    public static void main(String[] arguments){
        System.out.println( square(5) );
    }
}
Variable Scope

- Variables live in the block {} where they are defined (scope)

- Method input parameters are like defining a new variable in the method

```java
class SquareChange {
    public static void printSquare(int x) {
        System.out.println("printSquare x = " + x);
        x = x * x;
        System.out.println("printSquare x = " + x);
    }

    public static void main(String[] arguments) {
        int x = 5;
        System.out.println("main x = " + x);
        printSquare(x);
        System.out.println("main x = " + x);
    }
}
```

Note: this is for basic types ...
Methods

You should know this, but repetition is good 😊

- Big programs are built out of small methods
- Methods can be individually developed, tested and reused
- The user of method does not need to know how it works (this is called “abstraction”)

Conditionals

```
if (CONDITION) {
    STATEMENTS
} else {
    STATEMENTS 2
}
```

```
if (CONDITION) {
    STATEMENTS 1
} else if (CONDITION) {
    STATEMENTS 2
} else if (CONDITION) {
    STATEMENTS 3
} else {
    STATEMENTS 4
}
```
Conditions ...

Are a combination of:

- **Comparison operators:**
  - $x > y$, $x \geq y$: $x$ is greater than $y$, $x$ is greater or equal to $y$
  - $x < y$, $x \leq y$: $x$ is less than $y$, $x$ is less than or equal to $y$
  - $x == y$: $x$ equals $y$ (equality: ==, assignment: =)
  
  (NOT for Strings, non basic types or doubles)

- **Boolean operators:**
  - `&&`: logical AND
  - `||`: logical OR
  - `!`: logical NOT

- **Comparison operators:**
  - Do not call `==` for String
    String is an object, so Java will look at its memory (more later)
    Instead use `equals` (e.g., `s1.equals(s2)`)

  - Do not call `==` for doubles
    ```java
double a = Math.cos(Math.PI / 2);
double b = 0.0;
a = 6.123233995736766E-17
a == b will return FALSE
use Math.abs(a-b) < somethingSmall
```
Conditional Switch ...

switch (variable) {
    case value1:
        STATEMENTS
        break;
    case value2:
        STATEMENTS
        break;
    ...
    default:
        STATEMENTS
        break;
}

Exercise 2


Reminder:
- if (cond) {
  } else if (cond) {
  } else {
  }
- Condition operators: x > y, x >= y, x < y, x <= y, x == y
- Boolean operators: && (AND), || (OR), ! (NOT)
Loops (1)

```java
while (CONDITION) {
    do{
        STATEMENTS
    }
    while (CONDITION)
}
```

```java
int i = 0;
while (i < 3) {
    System.out.println("Counting #" + i);
    i += 1;
}
```

- Make sure that your loop has a chance to finish (especially if you cannot count a-priori)

Loops (2)

```java
for (initialization; CONDITION; update) {
    STATEMENTS
}
```

```java
for(int i = 0; i < 5; i=i+1) {
    System.out.println("Counting #" + i);
}
```

- Note: i = i+1 may be replaced by i++ (or ++i)
Loops (3)

Embedded loops

```
for(int i = 0; i < 5; i=i+1) {
    for (int j = 0; j < 7; j=j+1) {
        System.out.println(i + " " + j);
    }
}
```

- Scope of variables defined inside loop exist in the respective `for` block
- Scope of the iterating variable (e.g., `i,j`) defined in the initialization (also exist in the respective `for` block)

Loops (4)

Jumping loops (branching statements):

- You can terminate a loop by using `break`, this will move to the first instruction after the loop
- You can skip the current iteration of a loop using `continue`, which moves you to the next iteration

```
for(int i = 0; i < 5; i=i+1) {
    for (int j = 0; j < 7; j=j+1) {
        if (j == 3) continue; // go to j 4
        if (j == 5) break; // stop the inner loop
        System.out.println(i + " " + j);
    }
}
```
Exercise 3

Reminder:
- for (i=1; i <= 10; ++i) { ... }
- double a = (double) 1/2;

Exercise 4 - Homework

Reminder:
- for (i=1; i <= 10; ++i) { ... }
- System.out.print() prints in the same line
- System.out.println() adds a new line at the end
Java basics, Arrays

Arrays

- An array is an indexed list of values.

- You can make an array of any type
  - int, double, String, etc.

- All elements of an array must have the same type.
Arrays

Example: double[]

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5.0</td>
<td>1.5</td>
<td>2.33</td>
<td>0.0</td>
<td></td>
<td>-3.1</td>
</tr>
</tbody>
</table>

0 1 2 3 . . n-1
Arrays

- The index starts at zero and ends at length-1.

Example:

```java
int[] values = new int[5];
values[0] = 1; // CORRECT
values[3] = 1; // CORRECT
values[5] = 1; // WRONG!! compiles but
                // throws an Exception
                // at run-time
```

Arrays

- An array is defined as TYPE[].
- They are themselves just another type (more advanced)

```java
int[] values; // array of int

int[][] values; // array of int[]
               // (array of arrays)
```
Arrays

- Arrays of a specific size created using `new`:

```java
int[] values = new int[5];
```

- Or using a variable:

```java
int arraySize = 5;
int[] values = new int[arraySize];
```

- Or using `{}` to initialize the array values. This can only be used in the array declaration:

```java
int[] values = {2, 5, 3, 6, 6};
```

Array Access

- You can access elements in the array with `arrayName[index]`

**Example:**

```java
int[] values = {2, 5, 3, 6, 6};
values[2] = 12;  // assign value at position 2
                // {2,5,12,6,6}
int x = values[3]/2;  // read value at pos 3
                     // {2,5,3,6}
```
Array Access

- Arrays have a length variable built-in

Example:

```java
int[] values = new int[5];
int arraySize = values.length; // 5

int[] values2 = {2, 5, 3, 6, 6};
int arraySize2 = values2.length; // 5

public static void main (String[] arguments){
    if(arguments.length > 0)
        System.out.println("Passed "+
            arguments.length + " arguments");
}
```

Looping over Arrays

Examples:

```java
int[] values = {2, 5, 3, 6, 6};
for (int i=0; i<values.length; ++i)
    values[i]= values[i]*values[i];

int[] values2 = new int[5];
int i = 0;
while (i < values2.length){
    values[i] = i;
    ++i;
}
Exercise 5

Reminders/aid:

```java
int[] values = {2, 5, 3, 6, 6};
int[] values = new int[SIZE];
```

array[index] accesses value at index position, with index starting at 0

array.length gives the array size

```java
for (initialization; CONDITION; update) {
    STATEMENTS
}
```

You can get the max possible int with `Integer.MAX_VALUE;`

Common problems
Common problems (1)

- **Array Index vs Array Value**
  ```java
  int[] values = {2, 5, 3, 6, 6};
  System.out.println(values[0]); // 2
  ```

- **Curly braces {...} after if/else, while/for**
  ```java
  for (int i = 0; i < 5; i++)
      System.out.println("Hi");
  System.out.println("Bye");
  ```

Output?

Common problems (2)

- **Variable initialization**
  ```java
  int getMinValue(int[] vals) {
      int min = 0;
      for (int i = 0; i < vals.length; i++) {
          if (vals[i] < min) {
              min = vals[i];
          }
      }
  }
  ```

  What if vals = {1,2,3}?
  Set min = Integer.MAX_VALUE or vals[0]
Common problems (2)

- Defining method within method

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

- General comment:
  - Use `System.out.println` throughout your code

(aside, in our IDE)

- Ctrl-Shift-L: show list of keyboard shortcuts
- Ctrl-S: save
- Ctrl-Shift-F: aligns code
- Ctrl-O: opens an autocomplete window to jump to a member definition for all members of the open type
- Ctrl-Shift-O: organize imports automatically
- Ctrl-Shift-T: opens a window to open a type via autocompletion
- Ctrl-Shift-R: opens an autocomplete window for resources (files)
- F3: jump to definition

(On a Mac, replace Ctrl above with Cmd.)
Java resources

Many resources online ...

https://docs.oracle.com/javase/tutorial/
tutorialLearningPaths.html

« Thinking in Java » by Bruce Eckel (older versions of the book available online)

Exercises 4, 6, 7

Homework (together with anything else you did not finish in class)