ITI 1121. Introduction to Computing II

Java Memory Aid

by

Marcel Turcotte

Version January 9, 2020

Preambule

Preambule

Overview

Overview

Java Memory Aid

A concise overview the Java language focusing on types declarations, control structures and method calls. Object oriented programming, interface, and generics will be presented later.

General objective:

- By the end of this module, you should be able to write Java programs with a level of complexity similar to those from the course ITI1120.

Preambule

Learning objectives

Learning Objectives

- **Name** Java primitive types
- **Use** Java predefined types
- **Declare** variables
- **Develop** applications that are using arrays
- **Use** control structures to solve problems
- **Decompose** large problems into smaller ones to create structured applications

Lectures:

- Koffman & Wolfgang Appendice A (pages 541-555)

Types

Primitive types

The table below lists the characteristics of the primitif types. These types are also predefined. In class, we discuss the differences between primitive and reference types.

Type	**Size**	**Maximum**	**Examples**
boolean	1		**true**,**false**
char	16	'\uFFFF'	'a', 'A', '1', '*'
byte	8	127	-128, -1, 0, 1, 127
short	16	32767	-128, -1, 0, 1, 127
int	32	2147483647	-128, -1, 0, 1, 127
long	64	9223372036854775807	-128L, 0L, 127L
float	32	3.4028235E38	-1.0f, 0.0f, 1.0f
double	64	1.7976931348623157E308	-1.0, 0.0, 1.0

Reference types

- **Reference types** are: **classes**, **interfaces**, **enum types**, or **arrays**.
- Reference types will be presented in later modules.

Type declaration

$$\overset{\text{type}}{\texttt{int}}\ \overset{\text{identifier}}{\texttt{age}}\,;$$

- One must declare the **type** of each **variable** and **parameter**, as well as that of the **returned value**. Above, we are declaring that the variable **age** is of type **int**.

Compile-time error: “cannot find symbol”

```java
public class Test {
    public static void main(String[] args) {
        age = 21;
    }
}
```

- In the above example, the variable **age** has not been declared.

```
Test.java:3: error: cannot find symbol
        age = 21;
        ^
  symbol:   variable age
  location: class Test
1 error
```

Solution

- One must declare the **type of the variable**, here **int** (line 3), before using the variable (line 4).

```
1 public class Test {
2     public static void main(String[] args) {
3         int age;
4         age = 21;
5     }
6 }
```

Type declaration: methods

$$\begin{aligned}
&\mathbf{public}\ \overset{\text{type}}{\mathbf{int}}\ \mathrm{sum}(\overset{\text{type}}{\mathbf{int}}\ a\ ,\ \overset{\text{type}}{\mathbf{int}}\ b)\ \{\\
&\qquad \mathbf{return}\ a+b\,;\\
&\}
\end{aligned}$$

- One must declare the **type** of each **parameter**, as well as that of the **returned value**.

Compile-time error: returned value and parameters

```java
public class Test {
    public sum(a, b) {
        return a+b;
    }
}
```

```
Test.java:2: error: invalid method declaration; return type required
    public sum(a, b) {
           ^
Test.java:2: error: <identifier> expected
    public sum(a, b) {
                ^
Test.java:2: error: <identifier> expected
    public sum(a, b) {
                   ^
3 errors
```

Type of the returned value: void

- Some methods **do not return a result**, this is the case for the method **swap** below, the type must then be **void** (returns nothing).

```
public static void swap(int[] xs) {
    int tmp;
    tmp = xs[0];
    xs[0] = xs[1];
    xs[1] = tmp;
}
```

Variables

Definition: scope

*The **scope** of a declaration is the region of the program within which the entity declared by the declaration can be referred to using a simple name*

The Java Language Specification,
Third Edition, Addison Wesley, p. 117.

Definition: scope of local variables en Java

*The **scope of a local variable declaration** in a block is the rest of the block in which the declaration appears, starting with its own initializer and including any further declarators to the right in the local variable declaration statement*

The Java Language Specification,
Third Edition, Addison Wesley, p. 118.

$\Rightarrow$ A.K.A. static or lexical scope

Definition: scope of a parameter Java

The ***scope of a parameter*** *of a method or constructor is the entire body of the method or constructor*

The Java Language Specification,
Third Edition, Addison Wesley, p. 118.

⇒ A.K.A. static or lexical scope

Arrays

Reference to an array

```
   type        identifier
int []           xs ;
```

- **Declaring** a **reference variable** pointing to an **array**.
  - The syntax [] is used to declare an array. The type that precedes the brackets is type for each element of the array.
  - In the example above, **xs** is a reference to an array of integers.
  - **Important.** Declaring a reference to an array **does not create the array**. Only the reference variable*.

*Consult the module on reference types

Creating an array

$$\text{xs} = \textbf{new}\ \overset{\text{type}}{\textbf{int}}\,[\,\overset{\text{size}}{5}\,];$$

- **Creating** an array of integers of size **5**.
  - **Arrays** are objects.
  - The keyword **new** is used to create an object (which is created during the execution of the program).
  - The type of the object is **array of integers**.
  - The **size** of the array is 5.
  - The reference of the array is saved in the referencee variable named **xs**.

Accessing the content of an array

$$\overset{\text{index}}{}$$

```
value = xs[0];
```

- **Accessing** the content of an array.
  - One uses the name of the **reference variable** followed by **index** surrounded by square brackets.

Declaring, creating, and accessing the content of an array

- The method **main declares** a local variable of type reference to an array of integers (line 3), **creates** an array of integers of size 5 (line 4), and **assigns** the value 100 to the position 0 of the array designed by **xs** (line 5).

```
1 public class Test {
2     public static void main(String[] args) {
3         int[] xs;
4         xs = new int[5];
5         xs[0] = 100;
6     }
7 }
```

Initializing an array

- Line 4 shows how to **create** and **initialize** an array with values.

```
1 public class Test {
2     public static void main(String[] args) {
3         int[] xs;
4         xs = new int[] {0, 1, 2, 3, 4, 5};
5     }
6 }
```

- **Declaring** a reference variable for a **two-dimensional array** (line 2).
- **Creating** a two-dimensional array (line 3).
- **Assigning** a value to one of the cells of a two-dimensional array (lines 8 and 10).

```
 1  int size = 4;
 2  double[][] m;
 3  m = new double[size][size];
 4
 5  for (int i=0; i<size; i++) {
 6      for (int j=0; j<size; j++) {
 7          if (i==j) {
 8              m[i][j] = 1.0;
 9          } else {
10              m[i][j] = 0.0;
11          }
12      }
13  }
```

Multidimensional arrays

- A **multidimensional array** is a one dimensional array where each cell is a reference to an other array (that could also be an array of references to other arrays) [line 5].

```
1  double[][] m;
2  m = new double[4][];
3
4  for (int i=0; i<m.length; i++) {
5      m[i] = new double[4];
6      for (int j=0; j<m[i].length; j++) {
7          if (i==j) {
8              m[i][j] = 1.0;
9          } else {
10             m[i][j] = 0.0;
11         }
12     }
13 }
```

Multidimensional arrays

- Knowing that a **multidimensional array** is a one dimensional array such that each cell contains a reference to an other array, we can create a **triangular matrix** and thus
- The example below shows a two dimensional array.

```
double[][] m;
m = new double[][] {{1.0},{0.0, 1.0},{0.0, 0.0, 1.0}};
```

Expressions

Arithmetic operators

Priority	Operator	Description
2	++, --	Increment, decrement, post-fix
3	++, --	Increment, decrement, pre-fix †
3	+, -	Sign unary
4	*, /, %	Multiplication, division, modulo
5	+, -	Addition, subtraction

† Associativity **right-to-left** for the pre-fix operators.

Exemples:

```
int i=0, sum, a=4, b=6, rest;
i++;
sum = a + b;
rest = sum % 2;
```

Logical operators

Priority	Operator	Description
3	!	Negation †
12	&&	Logical And
13	||	Logical Or

† Associativity `right-to-left` for the pre-fix operators.

Exemple:

```
if (! hasPrize && ! isSelected) {
    isOpen = true;
}
```

Relational operators

Priority	Operator	Description
7	<, <=	Less than, less than or equal
7	>, >=	Greater than, greater than or equal
8	==, !=	Equal, not equal

Examples:

```
if (age < 3) {
    price = 0.0;
} else if (age <= 13 || age >= 65) {
    price = 11.99;
} else {
    price = 14.50;
}
```

Statements

Statements

- **Each statement** is terminated by a **semi-colon** (;)
- The **empty statement** comprises only the **semi-colon** (;)
- **Zero or more statements** can be grouped to form a **bloc** using parentheses. Such bloc can be used everywhere a statement can be used.

Statements: body of a method

```
public int sum(int a, int b) {
      int value;
      value = a+b;
      return value;
}
```

- The **body of a method** is a **bloc of statements**

Statement: if

```
if (age >= 18) {
    voters = voters + 1;
    System.out.println("Can vote");
}
```

- If the true-branch of an **if** statement comprises more than one statement, a **bloc of statements** must be used.
- It is **recommended** to always use a bloc with the branches of the if statement.

Statement: if-else

```
if (100.0*exams/65.0  <  50.0) {
    grade = 100.0*exams/65.0;
} else {
    grade = laboratories + assignments + exams;
}
```

- If the false-branch of an **if-else** statement comprises more than one statement, a **bloc of statements** must be used.
- In the above example, the parentheses are not needed, but it is **recommended** to put them.

Statement: for

```
               initialization     test     increment
for (int i=0; i<10; i++) {
    System.out.println(i);
}
```

- The **for** loop comprises three expressions: **initialization**, **test**, and **increment**.
- It is **recommended** to declare the type of the variable controlling the loop within the initialization of the loop to limit the scope of the variable to the body of the loop only.

Statement: while

```
int i;
i = 0;

while (i<10) {
     System.out.println(i);
     i++;
}
```

- The **while** statement.
  - The body of the loop is executed as long as the condition is true, here, as long as **i<10** is true.
- The code produces the same result as the previous example (**for** statement).

Statement: return

```java
public static int indexOf(String word, String[] words) {
    for (int i=0; i<words.length; i++) {
        if (word.equals(words[i])) {
            return i;
        }
    }
    return -1;
}
```

- All the methods returning a value, i.e. all the methods, except those where the type of the returned value is void, must have at least one **return** statement.
- When executing the **return**, the method terminates immediately and it returns the argument of the return statement.

Statement: return

```
1  public class ArrayUtils {
2      public static int indexOf(String word, String[] words) {
3          for (int i=0; i<words.length; i++) {
4              if (word.equals(words[i])) {
5                  return i;
6              }
7          }
8          return -1;
9      }
10 }
```

- In the above example, if the given word is found in the array, then the statement on line 5 will be executed. The method terminates immediately and returns the index of the given word in the array. If the given word is not found, then, the statement on line 8 is executed and the value -1 will be returned indicating that the given word was not found in the array.

Calling a (class) method: ArrayUtils.indexOf("charlie",words)

```java
public class Test {

    public static void main(String[] args) {

        int result;

        String[] words;
        words = new String[] {"alpha", "bravo", "tango"};

        for (int i=0; i<words.length; i++) {
            result = ArrayUtils.indexOf(words[i],words);
            System.out.println(result);
        }

        result = ArrayUtils.indexOf("charlie",words);
        System.out.println(result);
    }

}
```

Main method: main

- A Java program comprises one or many classes, but generally many classes (and accordingly many files).
- The example on previous slides comprises two classes: **ArrayUtils** and **Test**.
- One of the two classes comprises a main method.
  - Its signatures is always as follows:

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

  - The main method is always the first method to be called (except for Applets and Web Applications, but these are not seen in ITI1121).

Scanner

Reading data from the user

Reading data in Java often requires using several classes and knowing about exceptions.

- For simple applications, one can use the class **Scanner**.

`https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html`

Reading data from the user

- One needs to pass as an argument to the object **Scanner** the reference of an object associated with the keyboard, **System.in**.

```
sc = new Scanner(System.in);
```

- The object **Scanner** has many methods, namely a method to read the next integer, **nextInt**, the next floating point number, **nextDouble**, or the next line, **nextLine**.

https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html

```java
import java.util.Scanner;
import static System.out;

public class Test {
    public static void main(String[] args) {

        Scanner sc; int age; String answer;

        sc = new Scanner(System.in);

        out.print("How old are you? ");
        age = sc.nextInt();

        sc.nextLine();

        out.println("What is your favorite color?");
        answer = sc.nextLine();

        out.print("You are " + age + " years old");
        out.println(" and you like the color " + answer);

    }
}
```

Convention

Writing conventions

- The name of the variables, attributes, and methods starts with a **lower case** letter: **door**.
- Whenever a symbol consists of **many words**, the first letter of each word is a **capital** letter: **pickOneDoor**.
- The name of the classes, interfaces, and enum types starts with an **upper case** letter: **Statistics**. Camel case is used whenever the symbol consists of more than one word: **CamelCase**.
- The name of the constants (**final**) is made of all **upper case** letters, the words are separated by the underscore symbol: **MAX_VALUE**.

package, qualified names and import

In Java, classes are grouped together to create **packages**.

In order to access to the elements of a package, the are two mechanisms:

- Use the fully qualified name.
- Use the **import** directive.

Fully qualified name

In the example below, one wishes to use the class **Random** from the package **java.util**. In order to do so, we use the fully qualified name for the type and to create the object.

```
public class TestFullyQualifiedName {

    public static void main(String[] args) {

        java.util.Random r;
        r = new java.util.Random();
        System.out.println(r.nextInt(10));

    }
}
```

import

In order to simplify the code, one can import a name and make it available for the compilation unit using the directive **import**.

```java
import java.util.Random;

public class TestImport {

   public static void main(String[] args) {

            Random r;
            r = new Random();
            System.out.println(r.nextInt(10));

      }
}
```

Prologue

Summary

- In Java, one must **declare the type** of each **variable**. For the methods, the type of each **parameter** and **returned value** must be declared.
- We have seen the syntax to **declare** reference variable to an array, **creating** and array, and **accessing** the values of an **array**.
- Each statement ends with the **semi-colon** (;).
- Statements are grouped together to form a **bloc** using **parentheses**.
- The main **operators** and **control structures** have been presented.

We did not discuss the following concepts

- **Visibility modifiers**: **public** and **private**.
- The keyword **static** associated with **class variables and methods**.
- The **dot notation**: **words.length**.

These concepts will be presented in the **object oriented programming** module.

Prerequisites

Appendix: Getting started with Java.

Getting Java

Where do I get Java to install it on my machine?

- You can grab your copy of the Java Platform, Standard Edition (Java SE) from
  - `http://www.oracle.com/technetwork/java/javase/downloads/index.html`.

Prerequisites

Source Code Editor

Editing your code

You will also need an editor, such as

- Atom
  (macOS, Windows, Linux, `https://atom.io`)
- Notepad++
  (Windows, `http://notepad-plus-plus.org`)
- Sublime Text
  (macOS, Windows, Linux, `http://www.sublimetext.com`)
- TextWrangler
  (macOS, `http://www.barebones.com/products/textwrangler/`)
- TextMate
  (macOS, `http://macromates.com`)

Prerequisites

Command prompt

Command/Shell

You need to **navigate your file system**.

- In the labs: “Start”, then “All Programs”, then“Programming”, then “Java Development Kit”, then “Java-Cmd-xxxx”
- On Windows:
  - use the“Start" Menu, select “Run” and type “cmd”
  - Change directory: “cd DirectoryName”
  - List directory content: “dir”

Getting started

Getting started

Simplest Class

Simple compilable class

Here is a simple class:

```java
public class Work {

}
```

Save it as “Work.java”. Move to the corresponding directory and compile:

```
> javac Work.java
```

It worked. But, we cannot run it:

```
> java Work
Error: Main method not found in class Work, please define
the main method as:
   public static void main(String[] args)
```

Getting started

“main” method

Adding a main() method

We need to have a **main** method, a starting point.

```java
public class Work {

  public static void main(String[] args) {

  }
}
```

Now we can **run** it:

```
> javac Work.java
> java Work
```

It worked. But it does nothing ...

Getting started

Printing something

Printing something

We can **print** something:

```java
public class Work {

  public static void main(String[] args) {

    System.out.println("yes!");

  }
}
```

```
> javac Work.java
> java Work
yes!
```

Variables and Methods

Variables and Methods

Primitive Data Types

Primitive data types

Primitive	Range	Size	Wrapper
byte	$-128 \dots 127$	8	Byte
short	$-32,768 \dots 32,767$	16	Short
int	$-2^{31} \dots 2^{31} - 1$	32	Integer
long	$-2^{63} \dots 2^{63} - 1$	64	Long
float	roughly $\pm 10^{-38} \dots \pm 10^{38}$	32	Float
double	roughly $\pm 10^{-308} \dots \pm 10^{308}$	64	Double
boolean	“true” and “false”	1 (8?)	Boolean
char	Unicode character set	16	Character

Variables and Methods

Primitive Variables

Primitive variables

Let's add some (**primitive**) variables in our code

- Declaring a variable:

```
long myLong;
```

- Declaring and initializing a variable:

```
byte i = 33;
```

- Accessing, modifying variables:

```
int myInt = 5;
int myOtherInt = myInt;
myOtherInt = myOtherInt + 5;
myInt++;
```

Primitive variables

```java
public class Work {
  public static void main(String[] args) {
    int myInt = 0;
    long myLong;
    boolean myBool = false;
    char myChar = 'c';

    myLong = 3L;
    System.out.println("myInt: " + myInt + " myLong: " + myLong +
      " myChar:" +  myChar + " myBool: " + myBool);
  }
}
```

```
> javac Work.java
> java Work
myInt: 0 myLong: 3 myChar:c myBool: false
```

Variables and Methods

Strings

Using Strings

```
public class Work {
  public static void main(String[] args){
    String s = "this is a string";
    String s2 = "this is another string.";
    String s3 = s+s2;
    s=s.concat(". ");
    String s4 = s2.substring(8,15);
    System.out.println(s);
    System.out.println(s2);
    System.out.println(s3);
    System.out.println(s4);
    System.out.println("s contains \"string\": " +
      s.contains("string") );
    System.out.println("s contains \"strong\": " +
      s.contains("strong") );
  }
}
```

Using Strings

```
> javac Work.java
> java Work
this is a string.
this is another string.
this is a stringthis is another string.
another
s contains "string": true
s contains "strong": false
>
```

See `https://docs.oracle.com/javase/8/docs/api/java/lang/String.html`

Variables and Methods

Adding and Calling a Method

Methods

```java
public class Work {

  /* This is a method */

  public static void do() {
    System.out.println("Hey, look!");
  }

  public static void main(String[] args) {

    do(); // calling do

  }
}
```

```
> javac Work.java
> java Work
Hey, look!
```

Syntax

Syntax

The “For” loop

Loop: for (;;) {}

```
public class Utils {

  public static void do() {
    for (int i=0; i < 10; i++) {
      System.out.println("Iteration : " + (i+1));
    }
  }
}
```

```
public class Work {
  public static void main(String[] args){
    Utils.do();
  }
}
```

Loop: for (;;) {}

```
> javac Work.java
> java Work
Iteration : 1
Iteration : 2
Iteration : 3
Iteration : 4
Iteration : 5
Iteration : 6
Iteration : 7
Iteration : 8
Iteration : 9
Iteration : 10
```

Syntax

The “While” loop

Loop: while () {}

```
public class Utils {
  public void do() {
    int i = 0;
    while (i<10) {
      System.out.println("Iteration : " + (i+1));
      i++;
    }
  }
}
```

```
public class Work {
  public static void main(String[] args) {
    Utils.do();
  }
}
```

Loop: while () {}

```
> javac Work.java
> java Work
Iteration : 1
Iteration : 2
Iteration : 3
Iteration : 4
Iteration : 5
Iteration : 6
Iteration : 7
Iteration : 8
Iteration : 9
Iteration : 10
```

Syntax

The “If.. Else” statement

Statement: if() {} else {}

```java
public class Utils {
  public void do(boolean formalParameter) {
    if(formalParameter) {
      System.out.println("I went through the \"if\" clause");
    } else {
      System.out.println("I went through the \"else\" clause");
    }
  }
}
```

```java
public class Work {
  public static void main(String[] args){
    Utils.do(false);
  }
}
```

Statement: if () {} else {}

```
> javac Work.java
> java Work
I went through the "if" clause
I went through the "else" clause
```

References I

E. B. Koffman and Wolfgang P. A. T.
***Data Structures: Abstraction and Design Using Java*.**
John Wiley & Sons, 3e edition, 2016.

D. J. Barnes and M. Kölling.
***Objects First with Java: A Practical Introduction Using BlueJ*.**
Prentice Hall, 4e edition, 2009.

P. Sestoft.
***Java Precisely*.**
The MIT Press, second edition edition, August 2005.

Marcel **Turcotte**
(`Marcel.Turcotte@uOttawa.ca`)

Guy-Vincent **Jourdan**
(`gjourdan@uOttawa.ca`)

School of Electrical Engineering and **Computer Science** (EECS)
University of Ottawa