Systems Software

COMP20081

Lecture 12 – Java Language Basics

Dr Michalis Mavrovouniotis
School of Science and Technology
ERD 200
Office Hours: Thursday 12:00-14:00

Recall and Lecture Overview

- Recall
  - Java Platform
- Overview
  - Java Primitive Data Types
  - Arrays and Strings
  - Objects
  - Passing Parameters

Some Terminology First

- Class
- Object or Instance
- Method or Function
- Parameters
- Arguments

```
public class Student {

  String name;

   public void setName(String n){
      name = n;
   }

   public String getName(){
      return name;
   }

   public static void main(String[] args) {
      Student bob = new Student();
      bob.setName(“Bob");
      System.out.println(bob.getName());
   }
```

Primitive Data Types

- Primitive type is predefined by the language and is named by a reserved keyword
- Java uses the same collection of primitive data types with C++
    - Integer types
    - Floating point types
    - Boolean type
    - Characters
- However, there are important differences
- Java support 8 primitive data types

Integer Types

- **byte**: an 8-bit integer $-2^7 \ldots 2^7 - 1$ (-128…127)
- **short**: a 16-bit integer $-2^{15} \ldots 2^{15} - 1$ (-32768…32767)
- **int:** a 32-bit integer $-2^{31} \ldots 2^{31} - 1$
- **long**: a 64-bit integer $-2^{63} \ldots 2^{63} - 1$

```
byte eight = 8;              //valid
short sixteen = 16;          //valid
int thirtytwo = 32;          //valid
long sixtyfour = 64L;        //valid
long sixtyfour = 64;         //not valid
```

Float Types

- **float**: a 32-bit floating point (saves memory especially with arrays)
- **double**: a 64-bit floating point

```
float thirdytwo = 32.0f;      //valid
float thirdytwo = 32.0;       //not valid
double sixtyfour = 64.0;      //valid
double sixtyfour = 64.0d;     //valid
```

Boolean Type

- **boolean** : 8-bits and takes true or false

```
boolean flag = 1;            //not valid
boolean flag = true;         //valid
bool flag = true;            //not valid
```

Char Type

- **char:** a 16-bit Unicode character
  - Unicode: “a computing standard for encoding, representing and handling text in most writing systems”.

```
char letterU =  U;          //not valid
char letterJ = ‘J’;         //valid
char letterB = “B”;         //not valid
char letterA = ‘A’;         //valid
char digit1 = ‘1’;          //valid
char digit0 = ‘0’;          //valid
```

Special Characters

- Some Java *escape sequences*:

Escape Sequence	**Meaning**
\b	backspace
\t	tab
\n	newline
\”	double quote
\’	single quote
\\	backslash

String

- Not a primitive data type
- It is supported by the **java.lang** package with the String class.
- Basically is an array of characters

```
import java.lang.String; or import java.lang.*;
```

```
String n = “Example”;
```

String (cont’d)

```
class StringTest {
   public static void main (String[] args) {
            String str1 = “espresso”;
            String str2 = “espresso”;
            System.out.println(str1.equals(str2));                          //true
            System.out.println(str1.toUpperCase());                         //ESPRESSO
            System.out.println(str1.toLowerCase());                         //espresso
            System.out.println(str1.substring(0,2));                        //es
            System.out.println(str1.startsWith(“o”));                       //false
            System.out.println(str1.endsWith(“o”));                         //true
            System.out.println(str1.replace(‘e’, ‘E’));                     //EsprEsso
   }
}
```

String Concatenation

- The string concatenation operator is (+).
- Appends one string to the end of another
- It can be also used to append other data types to a *string*

```
class University {
  public static void main (String[] args) {
        String name = “Nottingham Trent University ”;
        int year = 2018;

        /*part one*/
        System.out.println(name);
        System.out.println(“Nottingham ” + “Trent ” + “Univeristy”);
        /*part two*/
        System.out.println(“Nottingham\n” + “Trent\n” + “Univeristy\n”);
        /*part three*/
        System.out.println(“Nottingham Trent University ” + year);
        System.out.println(name + year);
        System.out.println(“Nottingham Trent University ” + 2018);
  }
}
```

String Concatenation (cont’d)

```
class Addition {
   public static void main (String[] args) {
            System.out.println(5 + 5);
   }
}
```

Correct output?

A. 55
B. 10
C. 5 + 5

String Concatenation (cont’d)

```
class Addition {
   public static void main (String[] args) {
            System.out.println(“5 plus 5 equals ” + 5 + 5 );
   }
}
```

Correct output?

A. 5 plus 5 equals 55
B. 5 plus 5 equals 10
C. 5 plus 5 equals 5 + 5

String Concatenation (cont’d)

```
class Addition {
    public static void main (String[] args) {
                System.out.println(“5 plus 5 equals ” + (5 + 5) );
    }
}
```

Correct output?

A. 5 plus 5 equals 55
B. 5 plus 5 equals 10
C. 5 plus 5 equals (5 + 5)

Operators

- Arithmetic operators
  - **+** Addition operator (also used for String concatenation)
  - **-** Subtraction operator
  - * Multiplication operator
  - **/** Division operator
  - **%** Remainder operator
- Equality and Relational Operators
  - **==** Equal to
  - **!=** Not equal to
  - **>** Greater than
  - **>=** Greater than or equal to
  - **<** Less than
  - **<=** Less than or equal
- Logical Operators
  - **&&** Conditional-AND
  - **||** Conditional-OR

Arrays

- Java arrays are considered as special kind of objects
- Therefore they have attributes (most important is length)
- Two way to define an array:

```
int[] anArray = {1,2,3,4};

int[] anArray = new int[4];   or int anArray[] = new int[4];
int anArray[0] = 1;
int anArray[1] = 2;
int anArray[2] = 3;
int anArray[3] = 4;

int len = anArray.length;
```

Multidimensional Arrays - 1

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

            String[][] names = new String[4][7];
   }
}
```

How can we get the length of the array?

Output

```
System.out.println(names.length);            //4
System.out.println(names[0].length);       //7
```

Multidimensional Arrays - 2

```
class ArrayExample {
   public static void main (String[] args) {
            String[][] names = {{“Dr. “ , “Mr. ” , “Ms. ”} , {“Smith” , ”Jones”}};

            System.out.println(names[0][0] + names[1][0]);
            System.out.println(names[0][1] + names[1][0]);
            System.out.println(names[0][2] + names[1][1]);
            System.out.println(names[0][0] + names[1][2]);
   }
}
```

What is the output of the program?

Output

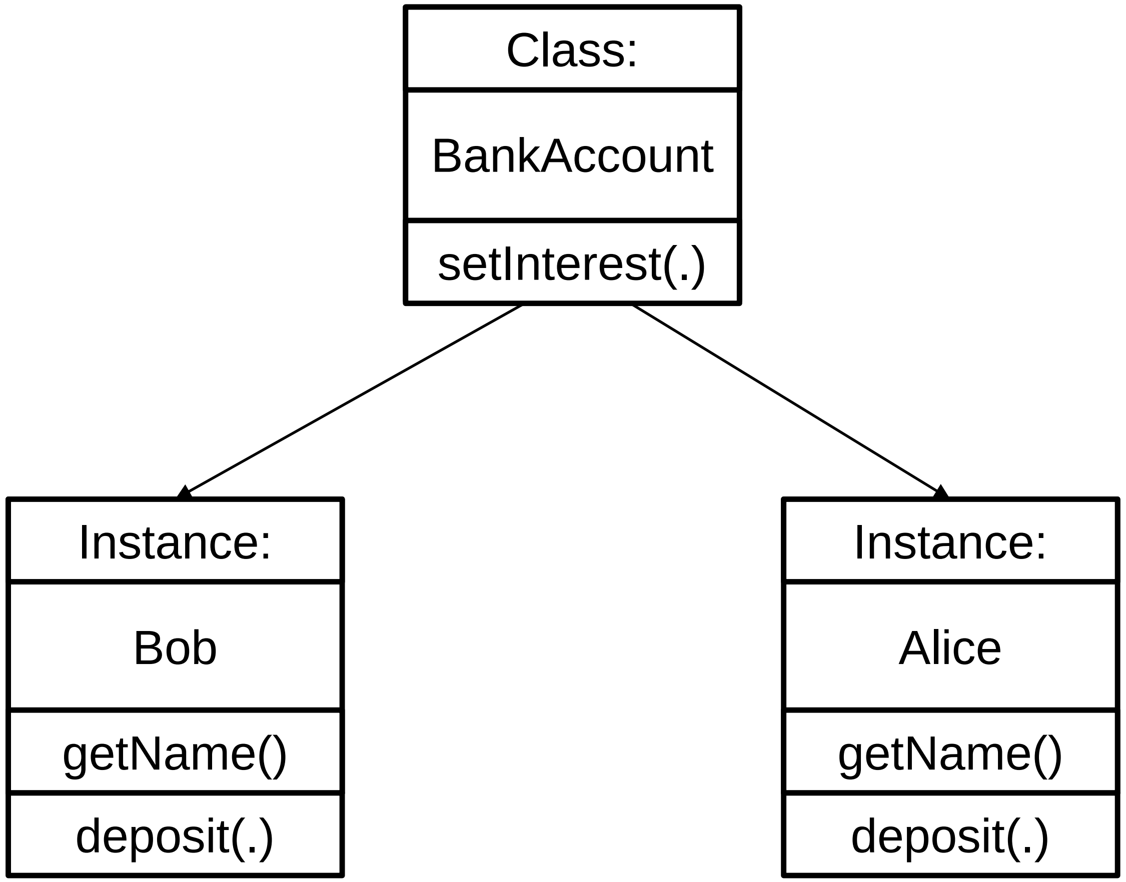
```
Dr. Smith
Mr. Smith
Ms. Jones
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 2
```

Methods

- Non-Static methods are known as **instance** methods
- Static methods are known as **class** methods

```
class Hello {
  public static void main (String[] args) {
        /*main method */
        public static void main(String[] args){
                greetingInstance("Bob");        //not legal
                greetingClass("Bob");           //legal
                Hello test = new Hello();
                test.greetingInstance("Bob"); //legal
                Hello.greetingClass("Bob");    //legal
        }
        /*class method */
        static void greetingClass(String name){
                System.out.println("Hello " + name);
        }
        /*instance method*/
        void greetingInstance(String name){
                System.out.println("Hello " + name);
        }
 }
```

Static vs Non-Static Methods



… more instances perhaps?

Static and Constant Parameters

- A static parameter is variable which is common to all instances of the class

```
static int interestRate;
```

- A constant is not variable and cannot be changed once initialized.
- Program will not compile if you try to change the value of a constant.

```
final int interestRate = 15;
final static int interestRate = 15;
```

Constants

- Constants are useful because:
  - They give meaning to otherwise unclear literal values
    - e.g., MAX_LOAD means more than the number 250 when reading code
  - If a constant is used in multiple places, its value need only to be modified in one place
  - Identify (to other programmers) that the value should not be changed!

Example of Passing Parameters

```
class PassingParameters {

   static void increase(int n) {
            System.out.println(“Number before increase ” + n);
            n++;
            System.out.println(“Number after increase ” + n);
   }

   public static void main (String[] args) {
            int number = 10;

            increase(number);
            System.out.println(“Number in main method: ” + number);
   }
}
```

Example of Passing Parameters – cont’d

```
class PassingParameters {

   ~~static void increase(int n)~~ {        //in C++  static void increase(int &n)
            System.out.println(“Number before increase ” + n);
            n++;
            System.out.println(“Number after increase ” + n);
   }

   public static void main (String[] args) {
            int number = 10;

            increase(number);
            System.out.println(“Number in main method: ” + number);
   }
}
```

Example of Passing Parameters – cont’d

```
class PassingParameters {

   static int increase(int n) {
            System.out.println(“Number before increase ” + n);
            n++;
            System.out.println(“Number after increase ” + n);

            return n;
   }

   public static void main (String[] args) {
            int number = 10;
            number = increase(number);
            System.out.println(“Number in main method: ” + number);
   }
}
```

Passing Parameters in Java

- In Java only the values are passed to the method for standalone primitives
- Whatever happens inside the method it does not affect the outside
- In Java only objects are passed by reference
- Therefore, if primitives data types are wrapped within an object then they will be passed by reference.

Summary

- Java fundamental primitive types
- Strings
- Arrays
- Classes and Instances
- Passing Parameters