1495 UNIVERSITY OF ABERDEEN

JC2002 Java Programming

Lecture 4: Java language basics

Basics of Java: topics

- In the remaining sessions today, we will cover the basics of Java programming language:
    - Basic Java console program structure.
    - Escape sequences, formatted printing, console input.
    - Primitive types, arithmetic operations, comparisons.
    - Conditional structures and loops.
    - Arrays and array lists.
- Much of the material is based on slides from ***Java: How to Program***, chapters 2, 4, 5, 7, available via MyAberdeen

Learning objectives

- After the theory sessions today, you should be able to:
  - Write simple Java console programs printing output to the console and receiving user input from the keyboard.
  - Use variables, basic arithmetic operations and comparisons.
  - Implement conditional structures and loops.
  - Define and initialize arrays and ArrayLists and use them for simple tasks, like computing the sum of the elements in an array.

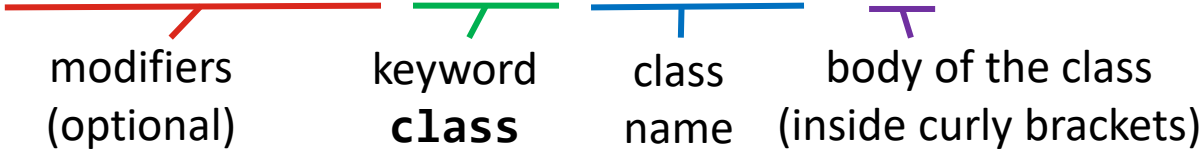
How to write a basic Java program?

- Java programs are organized as *classes* that include *methods*:
  - The basic rule is that the source code for a public Java class must be saved in a `.java` file with the same name as the public class.
  - Classes and methods will be explained in more detail later.
- Every Java program requires one *public* class with *public static* method **`main`** that serves as starting point of the program:
  - When program MyClass is started with command **`java MyClass`**, the program starts in method **`main`** of class **`MyClass`**

Declaring classes and methods

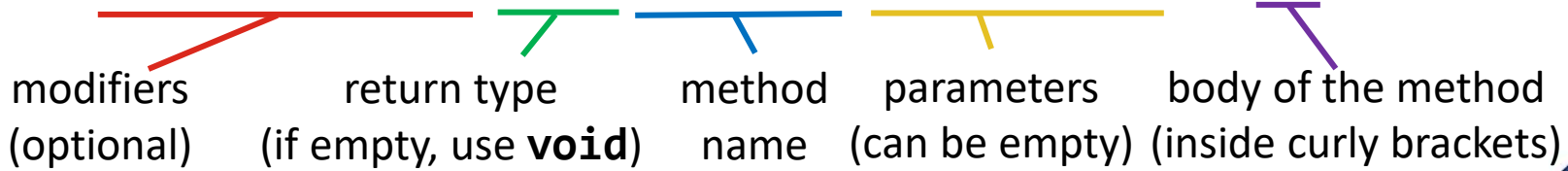
- The basic syntax for declaring a class:

```
public static class MyClass { … }
```



- The basic syntax for declaring methods (inside classes):

```
protected final void myMethod(int a, int b) { … }
```



Simple Java console program example (1)

```
1  // Example text-printing Java program Welcome1.java
2
3  public class Welcome1 {
4      // main method begins execution of Java application
5      public static void main(String[] args) {
6          System.out.println("Welcome to Java programming!");
7      } // end section main
8  } // end class Welcome1
```

Simple Java console program example (1)

```
1  // Example text-printing Java program Welcome1.java
2
3  public class Welcome1 {
4      // main method begins execution of Java application
5      public static void main(String[] args) {
6          System.out.println("Welcome to Java programming!");
7      } // end section main
8  } // end class Welcome1
```

You can add comments that are ignored by the compiler using **//** or **/* … */**

Simple Java console program example (2)

```
1  // Example text-printing Java program We
2
3  public class Welcome1 {
4      // main method begins execution of Java application
5      public static void main(String[] args) {
6          System.out.println("Welcome to Java programming!");
7      } // end section main
8  } // end class Welcome1
```

Every Java program requires at least one class

Method **`main`** is the starting point of the program

Simple Java console program example (3)

```
1  // Example text-printing Java program Welcome1.java
2
3  public class Welcome1 {
4      // main method begins execution of Java application
5      public static void main(String[] args) {
6          System.out.println("Welcome to Java programming!");
7      } // end section main
8  } // end class Welcome1
```

Built-in method **`System.out.println`** is used to print one line of text in the console

Simple Java console program example (4)

```
1   // Example text-printing Java program Welcome1.java
2
3   public class Welcome1 {
4       // main method begins execution of Java application
5       public static void main(String[] args) {
6           System.out.println("Welcome to Java programming!");
7       } // end section main
8   } // end class Welcome1
```

```
$ javac Welcome1.java
$ java Welcome1
Welcome to Java programming!
$
```

Compile the code in file `Welcome1.java`

Run the compiled program `Welcome1`

Output printed by the program

Simple Java console program example 2

```
1  // Example text-printing Java program Welcome2.java
2
3  public class Welcome2 {
4      // main method begins execution of Java application
5      public static void main(String[] args) {
6          System.out.print("Welcome to ");
7          System.out.println("Java programming!");
8      } // end section main
9  } // end class Welcome2
```

```
$ java Welcome2
Welcome to Java programming!
$
```

Simple Java console program example 3

```
1   // Example text-printing Java program Welcome3.java
2
3   public class Welcome3 {
4       // main method begins execution of Java application
5       public static void main(String[] args) {
6           System.out.println("Welcome\nto\nJava\nprogramming!");
7       } // end section main
8   } // end class Welcome1
```

```
% java Welcome3
Welcome
to
Java
programming!
$
```

Format output with printf

- Use `System.out.printf` method for printing output to the console
  - "`f`" means "formatted": `printf` displays *formatted* data
- The arguments are placed in a *comma-separated list*
- Calling a method is also referred to as *invoking* a method
- Java allows large statements to be split over many lines
  - However, cannot split a statement in the middle of an identifier or string

Arguments for printf

- Method `printf`'s first argument is a *format string*
  - May consist of *fixed text* and *format specifiers*
  - Fixed text is output as it would be by `print` or `println` method
  - Each format specifier is a placeholder for a value and specifies the type of data to output
- Format specifiers are a percent sign (**%**) followed by a character that represents the data type.
- For a string, **`%s`** is a placeholder, for an **`int`**, **`%d`** is a placeholder.
- Other placeholders: **`%f`** for floating point, **`%b`** for **`Boolean.`**

Declaring variables in Java

- In Java, variables need to be declared before using them.
- Basic syntax for declaring a variable with default value:

```
public int number;
```

modifiers (optional) data type variable name

- Basic syntax of declaring a variable and assigning it a value:

```
private double number = 5.8;
```

modifiers (optional) data type variable name assigned value

Primitive types in Java

- Primitive types are not derived from other data types

Type	Size (bits)	Value range
`boolean`	1	`true` or `false`
`char`	16	0 to 65535
`byte`	8	-128 to +127 (-2^7 to $+2^7-1$)
`short`	16	-32,768 to +32,767 (-2^{15} to $+2^{15}-1$)
`int`	32	-2^{31} to $+2^{31}-1$ (about $-2\cdot10^9$ to $+2\cdot10^9$)
`long`	64	-2^{64} to $+2^{64}-1$ (about -10^{19} to $+10^{19}$)
`float`	32	about (+/-) $1.4\cdot10^{-45}$ to $3.4\cdot10^{38}$
`double`	64	about (+/-) $4.9\cdot10^{-324}$ to $1.8\cdot10^{308}$

Formatted printing with input and integers

```
1   // Example formatted printing Java program with input
2   import java.util.Scanner; // needed for input
3   public class Addition {
4       public static void main(String[] args) {
5           Scanner input = new Scanner(System.in);
6           System.out.print("Enter first integer: "); // prompt
7           int number1 = input.nextInt(); // read first number
8           System.out.print("Enter second integer: "); // prompt
9           int number2 = input.nextInt(); // read first number
10          int sum = number1 + number2; // add numbers and store the result
11          System.out.printf("Sum is: %d%n", sum);
12      } // end section main
13  } // end class Addition
```

```
Enter first integer: 42
Enter second integer: 88
Sum is: 130
```

Imported classes

- By default, package **`java.lang`** is imported in every Java program; thus, classes in **`java.lang`** (such as **`System`**) are the only ones that don't need to be imported
- In the previous example, **`Scanner`** class is needed to enable a program to read data for use in a program
  - Data can come from many sources, such as the user at the keyboard or a file on disk
  - Before using a **`Scanner`**, you must create it and specify the source of the data

Binary overflow

- In some other programming languages (like C), variables may overflow their range of allocated bits
  - For example, for bytes, 127+1 results -128
- In Java, such situations are usually prevented and will give an error, but binary overflows can still occur for `int` and `long`. Be cautious with large numbers!

```
$ java Addition
Enter first integer: 2000000000
Enter second integer: 2000000000
Sum is: -294967296
```

Formatted printing of floats

```
1   // Example formatted printing Java program with input
2   import java.util.Scanner; // needed for input
3   public class Addition2 {
4       public static void main(String[] args) {
5           Scanner input = new Scanner(System.in);
6           System.out.print("Enter x: "); // prompt
7           float x = input.nextFloat(); // read first number
8           System.out.print("Enter y: "); // prompt
9           float y = input.nextFloat(); // read first number
10          float sum = x + y; // add numbers and store the result
11          System.out.printf("Sum is: %f%n", sum);
12          System.out.printf("Sum is: %.2f%n", sum);
13      } // end section main
14  } // end class Addition2
```

```
Enter x: 1.255
Enter y: 2.75
Sum is: 4.005000
Sum is: 4.01
```

Arithmetic operations in Java

Java operation	Operator	Algebraic expression	Java expression
Addition	+	$f + 78$	`f + 78`
Subtraction	-	$f - c$	`f - c`
Multiplication	*	bm	`b * m`
Division	/	x / y or $\frac{x}{y}$	`x / y`
Remainder	%	$r \bmod s$	`r % s`

Arithmetic precedence in Java

- Arithmetic operations in Java follow the standard precedence
  - **Multiplication**, **division**, and **remainder** (*, / ,%) are evaluated first. If there are several operators of this type, they are evaluated from *left* to *right*
  - **Addition** and **subtraction** (+, -) are evaluated next. Multiple operators of this type are evaluated from *left* to *right*.
  - **Assignment** (=) is evaluated last
- To improve readability and to avoid mistakes, you can use parenthesis in complex expressions.

Java equality and relational operators

Algebraic operator	Java operator	Example Java condition	Meaning
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

Comparisons and booleans example

```java
// Example comparison Java program
import java.util.Scanner; // needed for input
public class Comparison {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter x: "); // prompt
        int x = input.nextInt(); // read first number
        System.out.print("Enter y: "); // prompt
        int y = input.nextInt(); // read first number
        boolean isLarger = x > y; // compare if x is larger than y
        boolean isLess = x < y; // compare if x is less than y
        System.out.printf("x is larger than y: %b%n", isLarger);
        System.out.printf("x is less than y: %b%n", isLess);
    } // end section main
} // end class Comparison
```

```
Enter x: 5
Enter y: 10
x is larger than y: false
x is less than y: true
```

```
Enter x: 5
Enter y: 5
x is larger than y: false
x is less than y: false
```

```
Enter x: 10
Enter y: 5
x is larger than y: true
x is less than y: false
```

Questions, comments?