Programming in the Small I

Instructor: Musard Balliu, `musard@chalmers.se`

`http://www.cse.chalmers.se/~musard`

Recap from last lecture

1. Course overview
2. Mental landscape
3. Compilers and interpreters
4. A taste of JAVA

QUESTIONS?

Today's Plan

The basics of imperative programming:

1. SimpleIO library and JavaDoc
2. primitive types
3. variables
4. the assignment statment
5. blocks
6. the notion of **SCOPE**
7. Strings

Java programs

- A program is a sequence of instructions (statements) that a computer can execute to perform a task
- Java program: a program written in Java programming language
- Syntax: a well-defined set of rules that determine what is allowed (and what is not) to write in a program
- Semantics: the meaning of a program written in the languages
- Pragmatics: programming style and conventions

Primitive types

There are eight primitive types in Java:

1. byte (8 bits, range -2^7 to $2^7 - 1$, i.e. -128 to 127)
2. short (2 bytes, range -2^{15} to $2^{15} - 1$, i.e. -32768 to 32767)
3. int (4 bytes, range -2^{31} to $2^{31} - 1$)
4. long (8 bytes, range -2^{63} to $2^{63} - 1$)
5. float (4 bytes, single-precision IEEE 754 floating point)
6. double (8 bytes, double-precision IEEE 754 floating point)
7. boolean (size not defined, conceptually equivalent to one bit)
8. char (2 bytes, Unicode character)

These are not objects!

Literals

Literals are everything you can type into a program to represent a value.

For example, there are literals for each of types introduced in previous slide.

- Characters: $'c', '1', \cdots$
- Booleans: *true*, *false*
- Integers: $12, -22, \cdots$
- Floating point: $12.3, 0.33333333, \cdots$

Names and Identifiers

Names are a fundamental abstraction and it is important to understand their syntax, semantics and pragmatics.

Identifiers are basic names and can refer to classes, variables, subroutines, constants ecc.

There is a well-defined coding convention for names in Java (see related link in course webpage).

Names can be reserved words in Java, e.g. main, public, private, . . .

Variables

Variables are names used to refer to chunks of memory (they are a bit like addresses).

We need to know the structure of the chunk of memory a variable is supposed to refer to (e.g., how many bytes). Because of this, variables are introduced by *declarations*:

```
type variableName;
```

```
type variable1, variable2;
```

Variable names can contain letters, digits, and underscores, but no spaces, exclamation marks, etc. They cannot start with a digit.

Allowed expressions

The allowed expressions depend on the type of the variable (for example, arithmetical expressions are allowed for variables of type `int`, but not of type `boolean`).

Examples of expressions of type `int`: `42`, `x + y`, `x / y`, `x % y`.

Examples of expressions of type `double`: the above (including %!), plus decimal numbers `1.24`.

Examples of expressions of type `boolean`: `true`, `false`, `x && y`, `!x`, `x || y`.

The assignment statement

General form of an assignment statement:

```
variable = expression;
```

The assignment statement

The *state* of the computer is given by the set of variables and their values.

After an assignment statement, the value of the variable is the value of the expression used in the assignment.

Thus, in general, an assignment will change the state of the computer. Assignments are in fact the main way of changing the state.

Exercise

Write a Java program that computes the amount of interest that is earned on a user provided ammount of Swedish Kronor invested at an interest rate of 0.15 for one year. The interest and the value of the investment after one year are printed to standard output.

Cast

Some values of one type can be *cast*, i.e., converted to values of a different type.

For example, values of type `int` can be cast to `float`, and the other way around.

In general, if values of some type `A` can be converted to values of type `B`, then the converse also holds: values of type `B` can be cast to values of type `A`. Java calls such types *compatible*. Compatibility is a symmetric relation.

Some types are not compatible. For example, values of type `boolean` cannot be converted to `int`.

More on expressions

An expression is a piece of code that represents or computes a value

- Literals, variables and method calls are building blocks for expressions
- Bulid in classes such as Math contain several expressions
- Arithmetic expressions: $+, -, *, /, \%$
- Increment $(x++)$ and decrement $(x--)$
- Relational operators: $==, !=, <, >=$
- Boolean operators: $\&\&, ||, !$
- Conditional operator *bexp* ? *exp*1 : *exp*2

Exercise

Write a program that prompts the user to input 2 real numbers and prints the absolute value of their difference on the screen.

Precedence rules

Precedence rules determine the order of evaluation of expressions (unlesss you use parenthesis)

- Unary operators: $++, --, !$, unary $+, -$, type cast
- Multiplication $(*)$, division $(/)$, modulo %
- Addition $+$ and substraction $-$
- Increment $(x++)$ and decrement $(x--)$
- Relational operators: $<, >, <=, >=$
- Equality $(==)$ and Inequality $(!=)$
- Boolean And &&
- Boolean OR $||$
- Conditional operator *bexp* ? *exp*1 : *exp*2
- Assignment $=, +=, -=, *=, /=, \% =$

Scope

A variable can only be used, e.g., in an assignment statement, *after* it has been declared. That is, in the program text, the declaration of the variable has in general to preceede its use, in the usual (Western) order: from top to bottom and from left to right.

There are many qualifications and exceptions to this statement, which we shall examine in due course. The question of when in a program text a variable may be used is one of the most important matters in programming: it is the question of *scope*.

Sequencing statements

In some programming languages, the semicolon ; is not used to delimit the end of a statement, but rather to *sequence* two statements together.

In Java, we can interleave variable declarations and assignments, and we can also declare and initialize a variable “in one go”.

A semicolon by itself is a “do nothing” statement.

Assignment examples

```
public class Assign {
    public static void main(String[] args) {
        int n;  // n can be used until the closing
                // brace of main
        boolean b;
        // System.out.println(n); error!
        b = false;
        n = 300;
        char c; // c can be used from here to the end,
                // but not above
        c = 'x';
        n = (int) c; // char can be converted to int
        // n = (int) b; causes a compiler error
    }
}
```

Assignment exercises

1. Write a program that swaps the values of two variables.

Initial code
Solution

2. Assuming the variables are of type `int` (and not too large in absolute value), swap their values without using a temporary variable.

Initial code
Solution

Abbreviations of special assignments

```
public class Assign {
    public static void main(String[] args) {
        double x = 1.234;
        x++; // x = x + 1;
        x += 20; // works for all arithmetic operators
    }
}
```

Blocks

A sequence of statements can be grouped together in a block by putting it between { and }.

The group of statements is then treated a single, compound statement (we'll see that in a moment when it comes to conditionals).

An empty block {} is another way of writing a "do nothing" statement (just like the single semicolon).

Blocks and scope

The most interesting thing about blocks of statements is that variable declared inside a block are not visible outside the block.

Within a block, the usual rule applies: variables can be used after their introduction, until the end of the block.

Once all the statements in a block have been executed, all variables defined inside the block go *out of scope*. The computer memory used to store their values is free to be used for something else. We say that the variables are *garbage-collected*.

Blocks

```
public class Block {
    public static void main(String[] args)
    { // Start block main
        int n = 10;

        { // Start inner block
            int m = 5;
            println(m); // OK
            println(n); // OK
        } // End inner block

        // println(m); error!
        println(n); // OK
    } // End block main
}
```

Blocks

Exercise: write a program that swaps the values of two variables, using a temporary variable that will be garbage-collected after the values are swapped.

Initial code
Solution

Strings and String literals

-A String literal is a sequence of characters double quotes
-Special characters are represented using backslash notation
-A String type is not a primitive type (more on this later).

Operations on strings $s, s1$:

- $s.length()$: returns the number of characters in s
- $s.equals(s1)$: returns true if s and $s1$ are the same string
- $s.charAt(n)$: returns the char value at position n
- $s.substring(n, m)$: returns the substring between n and m
- $s.substring(n)$: returns the substring from position n
- $s.indexOf(s1)$: returns index within s of first occurrence of $s1$
- $s.trim()$: removes spaces from string s

Conversions

You can convert String or a character into a corresponding number as follows:

- $Integer.parseInt(s)$: returns the integer corresponding to s
- $Double.parseDouble(s)$: returns the double corresponding to string s
- $Character.getNumericValue(c)$: returns integer corresponding to char c
- $String.valueOf(c)$: returns the string representation of char c

Exercise

Write a Java program that reads a string of 10 characters from standard input and returns the number of digits occurring in the string.

Exercises

- Make sure you can compile and run all the examples from this lecture.

- Read sections 2.1, 2.2, 2.3.3, 2.5, and 2.6 of David Eck's book.