The first programmer

Augusta Ada, Countess Lovelace (1815 - 1852)
Image from:
http://www.theguardian.com/science/blog/2012/oct/13/tuesday-ada-lovelace-day-inspirational-women

CSC110
Getting started with Java

Have you...

- Found and logged into to conneX and located the CSC 110 site?
  - Do not yet have a Computer Science account?

    Then go to http://accounts.csc.uvic.ca and follow the instructions there.
- Reviewed the course outline?
  - You can also find a link to this at the Computer Science department website (www.csc.uvic.ca → Current Students → Undergraduate → Undergraduate Courses)
- Obtained the textbook and started chapter 1?

Announcements

- Labs start next week
  - in ECS 250
  - Please attend your registered lab section
- Assignment 1 will be posted soon – make sure to get started!
- Need extra help?
  - Instructor office hours
  - TA office hours
  - Lab time
  - Computer Science Assistance Centre

Next few topics

- Basic Java programs with "println" statements

- Gain familiarity with:
  - Strings – they will let our programs produce output!
  - Comments – they will make our code understandable
  - Types
  - Variables
  - Expressions

  $\left.\begin{array}{l}\text{Types}\\\text{Variables}\\\text{Expressions}\end{array}\right\}$ These let us do computation

This will give you the necessary background to for Lab 1 and Assignment 1!

What is programming?

Programming is the act of translating an **idea** for a solution into a **clear set of instructions/expressions** in a language that can be **interpreted by a computer**.

To be an **effective** programmer you need to know the language well, but you also need to be able to “speak” or “translate” the ideas into the language clearly and correctly.

What is programming?

- **program**: A set of instructions made by you to be carried out by a computer.

- **program execution or “running” a program**: The act of carrying out the instructions contained in a program by a computer.

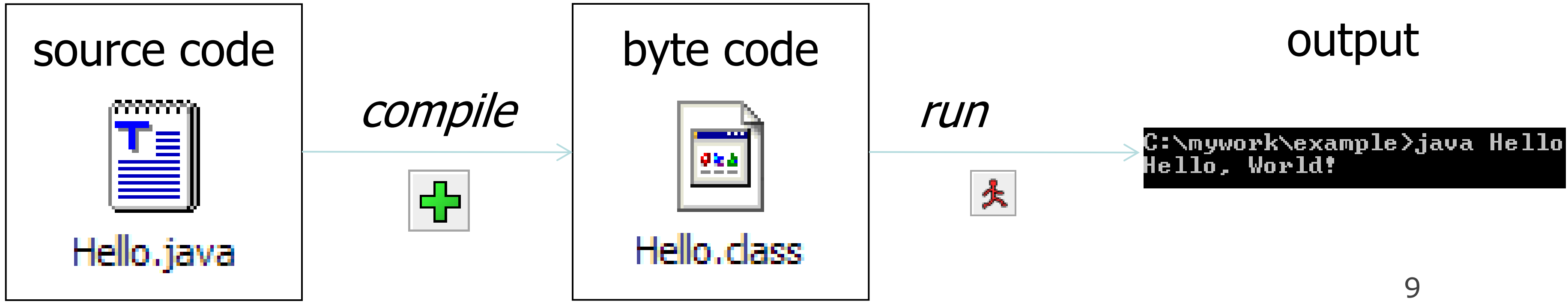
Basic Java programs with `println` statements

Compile/run a program

1. **Write** it.
   - **code** or **source code**: The set of instructions in a program.

2. **Compile** it.
   - **compile**: Translate a program from one language to another.
   - **byte code**: The Java compiler converts your code into a format named *byte code* that runs on many computer types.

3. **Run** (execute) it.
   - **output**: The messages printed to the user by a program.



```
C:\mywork\example>java Hello
Hello, World!
```

A Java program

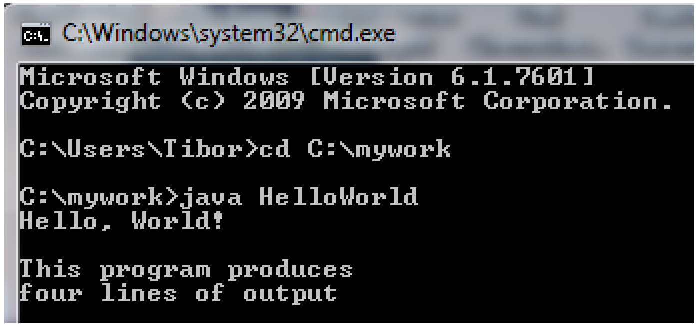
```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
        System.out.println();
        System.out.println("This program produces");
        System.out.println("four lines of output");
    }
}
```

- Its output:

```
Hello, World!

This program produces
four lines of output
```

- **console**: Text box into which the program's output is printed.



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.

C:\Users\Tibor>cd C:\mywork

C:\mywork>java HelloWorld
Hello, World!

This program produces
four lines of output
```

Structure of a Java program

```
public class name {
    public static void main(String[] args) {
        statement;
        statement;
        ...
        statement;
    }
}
```

class: a program

method: a named group of statements

statement: a command to be executed

- Every **executable** Java program consists of a **class**,
  - that contains a **method** named `main`,
  - and this contains the **statements** (commands) to be executed.

`System.out.println`

- A statement that prints a line of output on the console.

- Two ways to use `System.out.println` :

  - `System.out.println("`**`text`**`");`

    Prints the given message as output.

  - `System.out.println();`

    Prints a blank line of output.

Syntax

- **syntax**: The set of legal structures and commands that can be used in a particular language. Examples:
  - Every basic Java statement ends with a semicolon `;`
  - The contents of a class or method occur between `{` and `}`

- **Compiler error**: A problem in the structure of a program that causes the compiler to fail. Sometimes also called a **syntax error**
  - Missing semicolon
  - Too many or too few `{ }` braces
  - Illegal identifier for class name
  - Class and file names do not match

    ...

Syntax error example

```
1  public class Hello {
2      pooblic static void main(String[] args) {
3          System.owt.println("Hello, world!")_
4      }
5  }
```

- Compiler output:

```
Hello.java:2: <identifier> expected
    pooblic static void main(String[] args) {
           ^
Hello.java:3: ';' expected
}
^
2 errors
```

  - The compiler shows the line number where it found the error.
  - The error messages are, initially, tough to understand...
  - ... yet they will make more and more sense with experience!

Recall: Program structure

```
public class NameOfProgram
{
  public static void main(String [] args)
  {
      // Your program goes here
       System.out.println("Hello, world!");
  }
}
```

Strings

- **string**: A sequence of characters.
  - Starts and ends with a `"` quote `"` character.
    - The quotes do not appear in the output.
  - Examples:

    ```
    "hello"
    "This is a string.  It's very long!"
    ```

  - May not span multiple lines.

    ```
    "This is not
    a legal String."
    ```

  - (Later we'll see a way to have them span multiple lines – kind of.)
  - Note that we use straight quotes " not curly quotes ″

Escape sequences

- **escape sequence**: A special sequence of characters used to represent certain special characters in a string.

`\t`	tab character
`\n`	new line character
`\"`	quotation mark character
`\\`	backslash character

  - Example:

```
System.out.println("\\hello\nhow\tare \"you\"?\\\\");
```

  - Output:

```
\hello
how	are "you"?\\
```

Questions

- What is the output of the following `println` statements?

```
System.out.println("\ta\tb\tc");
System.out.println("\\\\");
System.out.println("'");
System.out.println("\"\"\"");
System.out.println("C:\nin\the downward spiral");
```

- Write a `println` statement to produce this output:

```
/ \ // \\ /// \\\
```

Answers

- Output of each `println` statement:

```
        a       b       c
\\
'
"""
C:
in      he downward spiral
```

- `println` statement to produce the line of output:

```
System.out.println("/ \\ // \\\\ /// \\\\\\");
```

Question

- What `println` statements will generate this output?

```
This program prints a
quote from the Gettysburg Address.

"Four score and seven years ago,
our 'fore fathers' brought forth on
this continent a new nation."
```

Answer

- What `println` statements will generate this output?

```
This program prints a
quote from the Gettysburg Address.

"Four score and seven years ago,
our 'fore fathers' brought forth on
this continent a new nation.‟
```

- `println` statements to generate the output:

```
System.out.println("This program prints a");
System.out.println("quote from the Gettysburg Address.");
System.out.println();
System.out.println("\"Four score and seven years ago,");
System.out.println("our 'fore fathers' brought forth on");
System.out.println("this continent a new nation.\"");
```

Question

- What `println` statements will generate this output?

```
A "quoted" String is
'much' better if you learn
the rules of "escape sequences."

Also, "" represents an empty String.
Don't forget: use \" instead of " !
'' is not the same as “
```

Answer

- What `println` statements will generate this output?

```
A "quoted" String is
'much' better if you learn
the rules of "escape sequences."

Also, "" represents an empty String.
Don't forget: use \" instead of " !
'' is not the same as "
```

- `println` statements to generate the output:

```
System.out.println("A \"quoted\" String is");
System.out.println("'much' better if you learn");
System.out.println("the rules of \"escape sequences.\"");
System.out.println();
System.out.println("Also, \"\" represents an empty String.");
System.out.println("Don't forget: use \\\" instead of \" !");
System.out.println("'' is not the same as \"");
```

Comments

- **comment**: A note written in source code by the programmer to describe or clarify the code.
  - Comments are not executed when your program runs.

- Examples:

```
// This is a one-line comment.

/* This is a very long
   multi-line comment. */
```

Using comments

- Where to place comments:
  - at the top of each file (a "comment header")
  - at the start of every method (seen later)
  - to explain complex pieces of code

- Comments are useful for:
  - Understanding larger, more complex programs.
  - Multiple programmers working together, who must understand each other's code.

Comments example

```
/* Tibor van Rooij, CSC 110, Fall 2015
   This program prints lyrics from a 1992 top-ten radio hit. */

public class TheCure {
    public static void main(String[] args) {
        // first part
        System.out.println("I don't care if Monday's blue");
        System.out.println("Tuesday's grey and Wednesday too");
        System.out.println("Thursday I don't care about you");
        System.out.println("It's Friday I'm in love");

        // second part
        System.out.println("Monday you can fall apart");
        System.out.println("Tuesday, Wednesday break my heart");
        System.out.println("Thursday doesn't even start");
        System.out.println("It's Friday I'm in love");    }
}
```

Data types

- **type**: A category or set of data values.
  - Constrains the operations that can be performed on data
  - Many languages ask the programmer to specify types for the data they will be using in their program

- Examples:
  - integer
  - real number
  - string

Java's primitive types

- **primitive types**: Eight simple types for numbers, text, etc.

Name	**Description**		**Examples**
`int`	integers	(up to $2^{31} - 1$)	`42, -3, 0, 926394`
`double`	real numbers	(up to 10^{308})	`3.1, -0.25, 9.4e3`
`char`	single text characters		`'a', 'X', '?', '\n'`
`boolean`	logical values		`true, false`

- (Also: `byte, short, long, float`)
- Why does Java distinguish integers vs. real numbers?

Expressions

- **expression**: A value or operation that computes a value.
  - Examples:
    ```
    1 + 4 * 5
    (7 + 2) * 6 / 3
    42
    ```
  - The simplest expression is a **literal value**.
  - A complex expression can use **operators** and **parentheses**.

Arithmetic operators

- **operator**: Combines multiple values or expressions.

`+`	addition
`-`	subtraction (or negation)
`*`	multiplication
`/`	division
`%`	modulus (a.k.a. remainder)

- As a program runs, its expressions are **evaluated**.
  - 1 + 1 evaluates to 2

  `System.out.println(3 * 4);` outputs `12`

  - How would we print the text `3 * 4` ?

- According to Java:

16 / 5   is   3

But why??

Integer division with `/`

- When we divide integers, the quotient is also an integer.

  `16 / 5` is `3`, not `3.2`

```
       3                    4
    ____                 ____
5 ) 16              10 ) 45
    15                   40
    --                   --
     1                    5
```

  – Dividing by 0 causes an error when your program runs.

Integer remainder with `%`

- The `%` operator computes the remainder from integer division.
  - `14 % 4`   is `?`
  - `218 % 5`   is `?`

```
        3                 43
    ______             _____
4 ) 14             5 ) 218
    12                 20
    __                 __
     2                  18
                        15
                        __
                         3
```

- Some possible uses of the `%` operator:
  - Obtain last digit of a number: `230857 % 10` is `7`
  - Obtain last 4 digits: `658236489 % 10000` is `6489`
  - See whether a number is odd: `7 % 2` is `1`, `42 % 2` is `0`

Precedence

- **precedence**: Order in which operators are evaluated.
  - Generally operators that are the same will evaluate left-to-right.

    `1 - 2 - 3` is `(1 - 2) - 3` which is `-4`

  - But `* / %` have a higher level of precedence than `+ -`

```
1 + 3 * 4            is 13

6 + 8 / 2 * 3
6 +   4   * 3
6 +     12           is 18
```

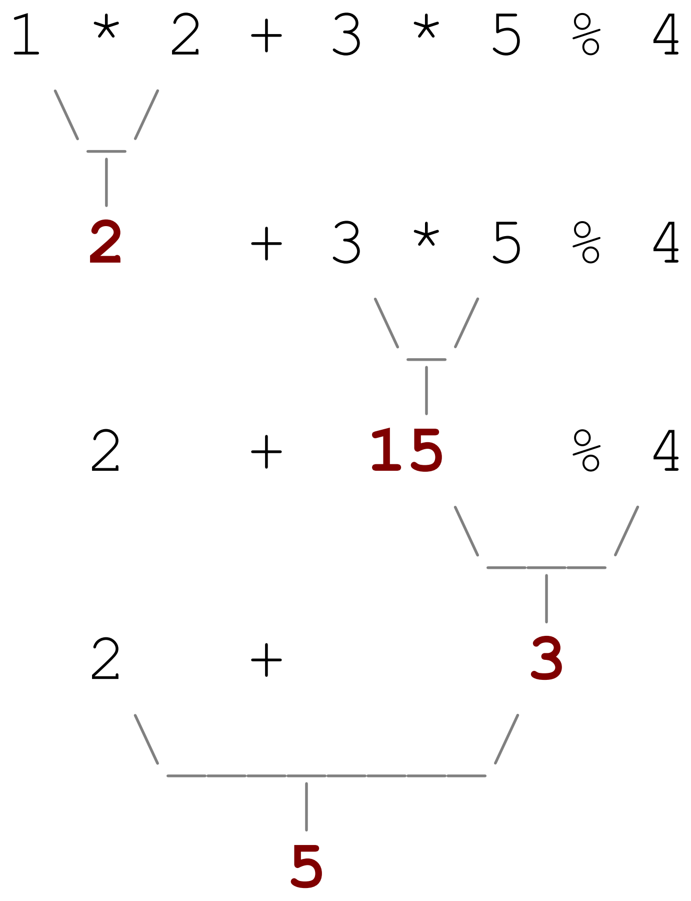
  - Parentheses can force a certain order of evaluation:

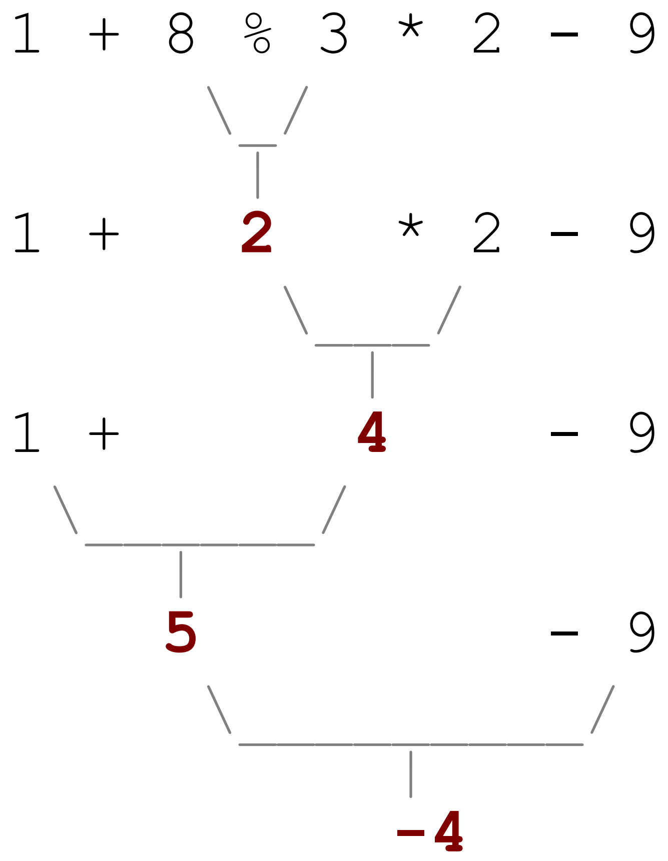
```
(1 + 3) * 4          is 16
```

  - Spacing does not affect order of evaluation

```
1+3 * 4-2            is 11
```

Precedence examples





These are called expression trees