Building Java Programs
Chapter 1

Introduction to Java Programming

Current Reading Assignment: Chapter 1

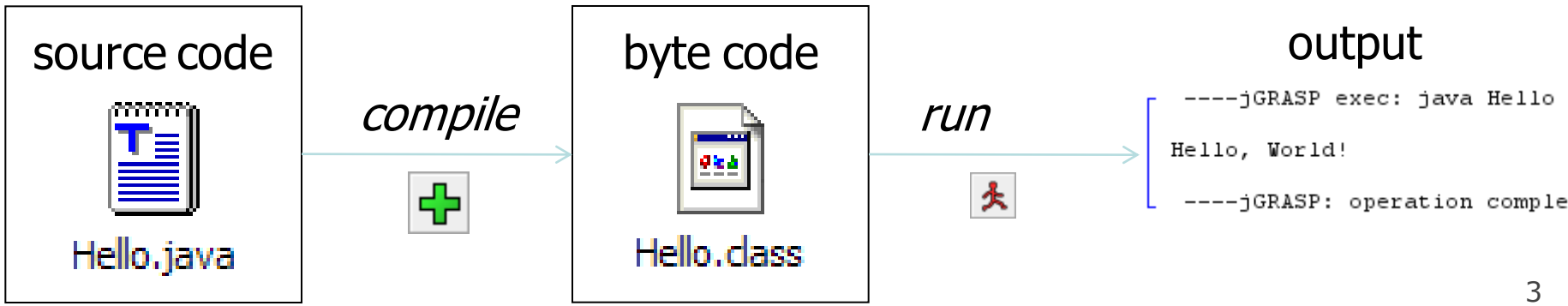
What is programming?

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 a *good* programmer you need to know the language well, but you also need to be able to "speak" the language clearly and correctly.

Compile/run a program

1. *Decide what to write.*
2. *Write* it.
   - **code** or **source code**: The set of instructions in a program.
3. *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.
4. *Run* (execute) it.
   - **output**: The messages printed to the user by a program.



A Java program

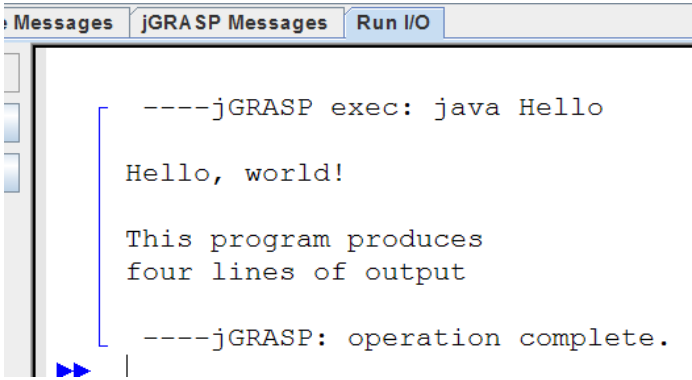
```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
        System.out.println();
        System.out.println("This program produces");
        System.out.println("four lines of output");
    }
}
```

- Its output:

```
Hello, world!

This program produces
four lines of output
```

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



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`,
    - that contains the **statements** (commands) to be executed.

Names and identifiers

- You must give your program a name.

```
public class GangstaRap {
```

  - Naming convention: capitalize each word (e.g. `MyClassName`)
  - Your program's file must match exactly (`GangstaRap.java`)
    - includes capitalization (Java is "case-sensitive")

- **identifier**: A name given to an item in your program.
  - must start with a letter or `_` or `$`
  - subsequent characters can be any of those or a number
    - legal: `_myName` `TheCure` `ANSWER_IS_42` `$bling$`
    - illegal: `me+u` `49ers` `side-swipe` `Ph.D's`

Keywords

- **keyword**: An identifier that you cannot use because it already has a reserved meaning in Java.

abstract	default	if	private	this
boolean	do	implements	protected	throw
break	double	import	**public**	throws
byte	else	instanceof	return	transient
case	extends	int	short	try
catch	final	interface	**static**	**void**
char	finally	long	strictfp	volatile
class	float	native	super	while
const	for	new	switch	
continue	goto	package	synchronized	

Syntax

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

- **syntax error** (**compiler error**): A problem in the structure of a program that causes the compiler to fail.
  - 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:

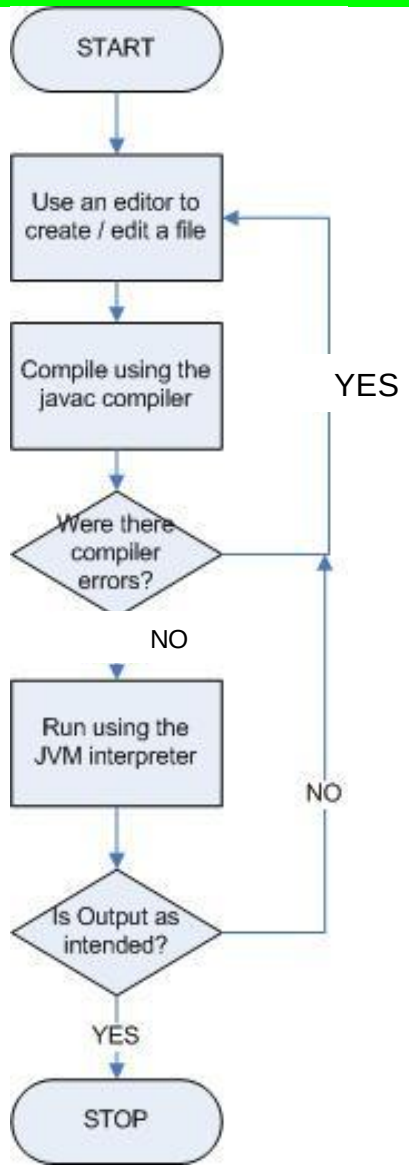
```
Hello.java:2: <identifier> expected
    pooblic static void main(String[] args) {
          ^
Hello.java:3: ';' expected
}
^
2 errors
```

  - The compiler shows the line number where it found the error.
  - The error messages can be tough to understand!

The Programming Process

The typical cycle for writing a Java program has four steps:

1. Edit — Open a text editor (any one will do). Enter Java language code. Save as a ".java" file.
2. Compile — Run the javac compiler to translate to bytecode. (Produces a ".class" file)
3. Run — Use the java (JVM) interpreter to execute the instructions in your program.
4. Check — Ensure the output is as you intended. Return to step 1 and make corrections if needed.



Strings

- **string**: A sequence of characters to be printed.
  - 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!"
```

- Restrictions:
  - May not span multiple lines.

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

  - May not contain a `"` character.

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

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("/ \\ // \\\\ /// \\\\\\");
```

Challenge Questions:

- What `println` statements will generate the following 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."
```

- What `println` statements will generate the following 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 "
```

15

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.

- Syntax:

```
// comment text, on one line
```

or,

```
/* comment text; may span multiple lines */
```

- 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

```
/* Suzy Student, CS 101, Fall 2019
   This program prints lyrics about ... something. */

public class BaWitDaBa {
    public static void main(String[] args) {
        // first verse
        System.out.println("Bawitdaba");
        System.out.println("da bang a dang diggy diggy");
        System.out.println();

        // second verse
        System.out.println("diggy said the boogy");
        System.out.println("said up jump the boogy");
    }
}
```