FCDS

Programming I

Lecture 1: Introduction to JAVA

Text Book

Building Java Programs: A Back to Basics Approach (2nd Edition) Stuart Reges & Marty Stepp;

Evaluation

Category	Percentage	Location	Date
Lab Assignments	*10%*	*In Lab*	*Weekly*
Mid Term	*20%*	*In Class*	*7th week*
Final Lab Exam	*20%*	*In Lab*	*13th week*
Final Exam	*50%*	*In Class*	*15th week*

Course Objectives

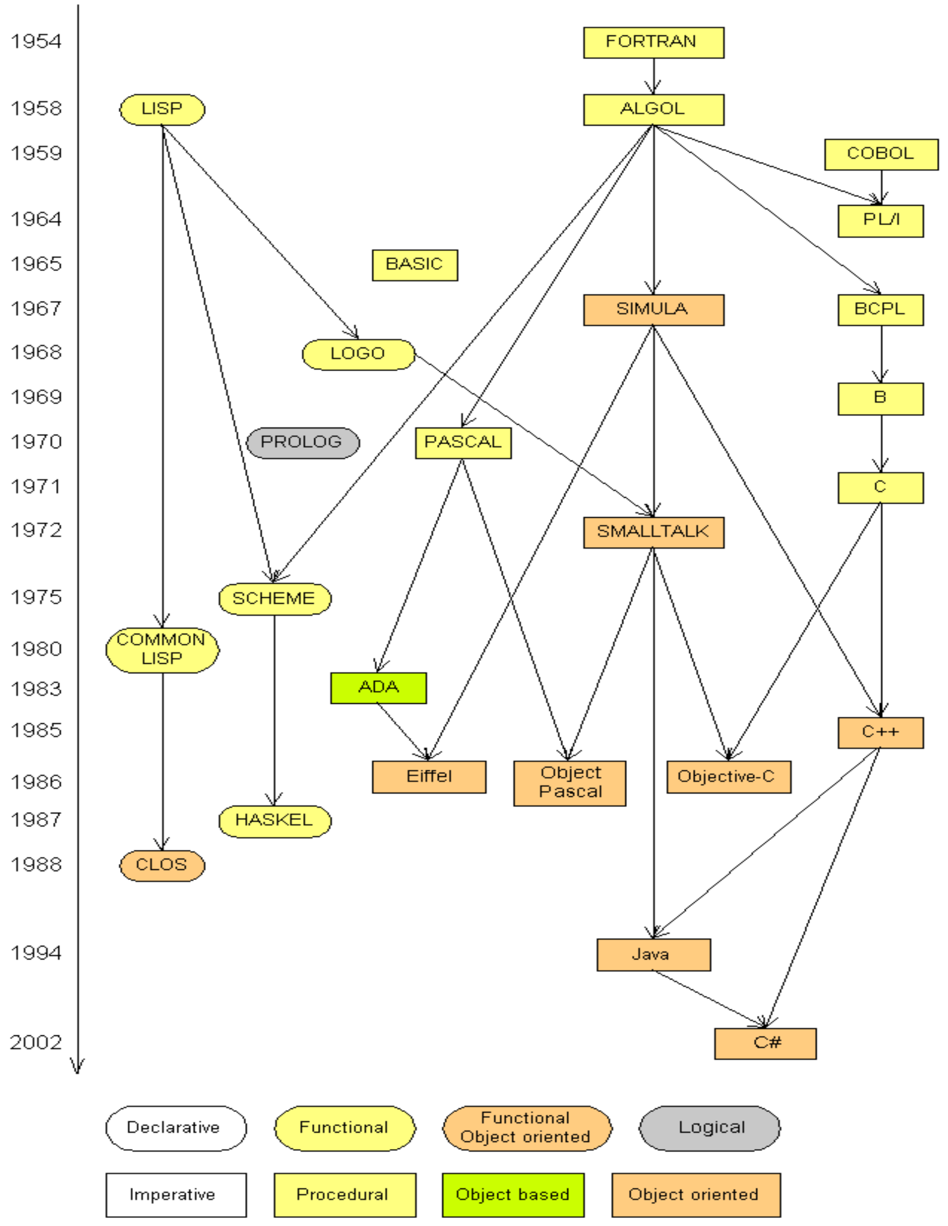
- Help students to understand the fundamentals of programming such as variables, conditional and iterative statements, methods, recursion, arrays, etc.
- Develop the student's ability to write a well-structured computer program to solve specified problems using ***Java***
- Teach students to use the ***Java*** SDK environment to create, debug and run simple ***Java*** programs

What is programming?

- **program**: A set of instructions to be carried out by a computer.
- **program execution**: The act of carrying out the instructions contained in a program.
- **programming language**: A systematic set of rules used to describe computations in a format that is editable by humans.
  - For example: ***Java***

Programming languages

- Some influential ones:
  - FORTRAN
    - science / engineering
  - COBOL
    - business data
  - LISP
    - logic and AI
  - BASIC
    - a simple language



Some modern languages

- *procedural languages*: programs are a series of commands
  - **Pascal** (1970): designed for education
  - **C** (1972): low-level operating systems and device drivers
- *functional programming*: functions map inputs to outputs
  - **Lisp** (1958) / **Scheme** (1975), **ML** (1973), **Haskell** (1990)
- *object-oriented languages*: programs use interacting "objects"
  - **Smalltalk** (1980): first major object-oriented language
  - **C++** (1985): "object-oriented" improvements to C
    - successful in industry; used to build major OSes such as Windows
  - **Java** (1995): designed for web apps/servers
    - Created by James Gosling and released by Sun microsystems
    - Runs on many platforms (Windows, Mac, Linux, cell phones...)

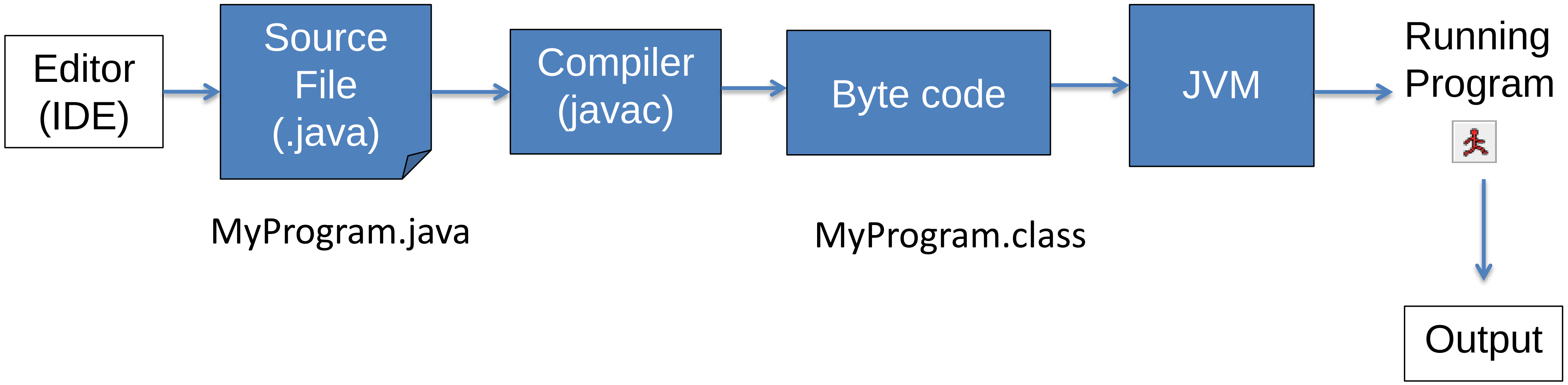
Process of Programming

- **Code**: describes program fragments (e.g., "*four lines of code*") or the act of programming (e.g., "Let' s code this into Java")
- The *process of execution* is often called **running**
- **Program** is stored in the computer as a series of binary numbers known as **machine language**
  - Machine language programs are **executable** programs
  - Modern programmers use **high-level programming languages** such as C++ and Java.
- High-level programming languages cannot be run directly on a computer
  - They first have to be **translated** into machine language
  - Translation happens using ***Compilers***

Process of Programming

- **Compiler**: software that often translates a program written in a programming language into an equivalent program in another computer language
  - Often but not always from a high level language into machine language
- Java programs compile into ***Java bytecodes (NOT machine lang.)***
  - *Intermediate* level (not as high as Java or as low as machine language)
  - One set of bytecodes can execute on many different machines
  - It represents the machine language of a *virtual* computer know as **Java Virtual Machine (JVM)**
- **Java Runtime Environment** (JRE): a program that executes Java bytecodes
- **Java development Kit** (JDK) = JRE + Java compiler.
- **Java Runtime Environment** (JRE) = JVM + Library Classes

Process of Programming



IDE (Editor)

- **Eclipse** is an integrated development environment (IDE).
  - Syntax highlighting editor
  - Easy to compile and execute programs
  - Debugging (finding and eliminating errors in the program)

Basic Java programs with *println* statements

A Java program

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

- **Its output:**

```
Hello, world!
```

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

Structure of a Java program

class: a program

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

Body of main method

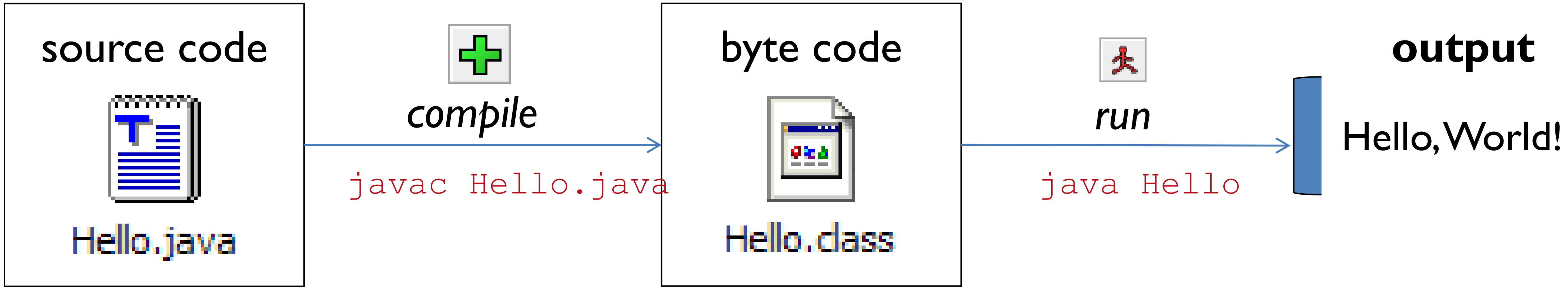
method: a named group of statements

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.

Compile/run a program

1. *Write it.*
   - **code** or **source code**: The set of instructions in a program.
2. *Compile it.*
   - **javac:** translates the program from Java to bytecode
   - **bytecode**: runs on many computer types (any computer with JVM)
3. *Run* (execute) it.
   - **output**: whatever the **programmer** instructs the program to do



System.out.println

- A **statement** that prints a line of output on the console.
  - pronounced "print-linn"
  - sometimes called a "println statement" for short

- 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.

Another Java program

```java
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
```

Names and identifiers

- You must give your program a name.

  ```
  public class MyClass {
  ```

  - Naming convention: capitalize each word (e.g. `MyClass`)
  - Your program's file must match exactly (`MyClass.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.out.println("Hello, world!")_
4      }
5  }
```

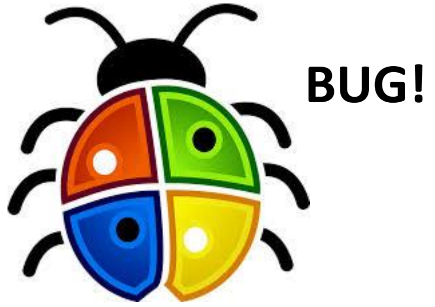
- Compiler output:

```
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!

Other types of Errors

- ***Logic errors:*** occur when you write code that doesn't perform the task it is intended to perform



- ***Runtime errors:*** are logic errors that are so severe that Java stops your program from executing.

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

Questions

- 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."
```

- 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 "
```

Answers

- `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.\"");
```

- `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.
- 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, FCDS, Fall 2020
   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");
    }
}
```