Programming in Java

Computer Science S-111
Harvard University

David G. Sullivan, Ph.D.

Programs and Classes

- In Java, all programs consist of one of more *classes*.
- For now:
  - we'll limit ourselves to writing a single class
  - you can just think of a class as a container for your program
- Example: our earlier program:

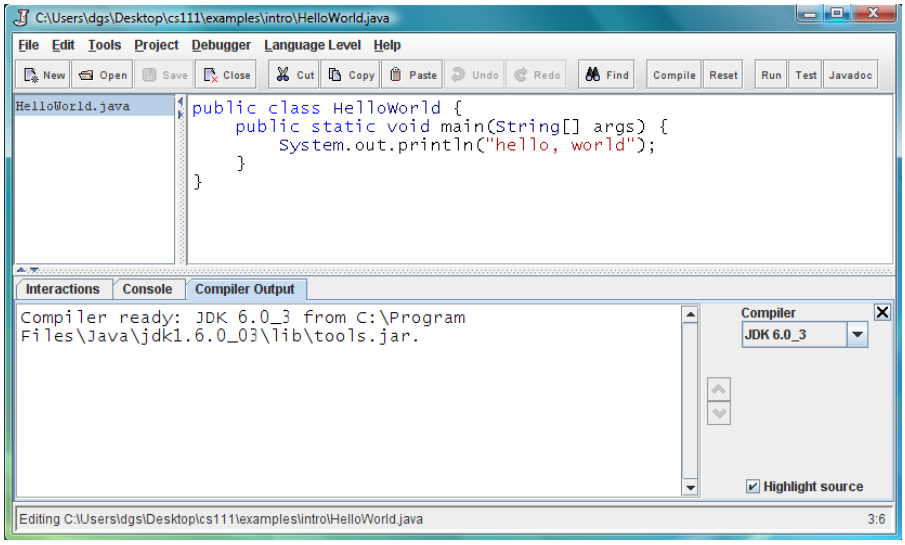
```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("hello, world");
    }
}
```

- A class must be defined in a file with a name of the form <classname>`.java`
  - for the class above, the name would be `HelloWorld.java`

Using an IDE

- An *integrated development environment (IDE)* is an application that helps you develop programs.
- We'll use the Dr. Java IDE.
  - PS 0 told you how to obtain and install it.
- With an IDE, you do the following:
  - use its built-in text editor to write your code
  - instruct the IDE to compile the code
    - turns it into lower-level instructions that can be run
    - checks for violations of the syntax of the language
  - instruct the IDE to run the program
  - debug as needed, using the IDE's debugging tools

Using Dr. Java



Format of a Java Class

- General syntax:

```
public class <name> {
    code goes here...
}
```

  where <name> is replaced by the name of the class.

- Notes:
  - the class begins with a *header*:

    `public class` <name>
  - the code inside the class is enclosed in curly braces ({ and })

Methods

- A method is a collection of instructions that perform some action or computation.
- Every Java program must include a method called `main`.
  - contains the instructions that will be executed first when the program is run
- Our example program includes a `main` method with a single instruction:

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("hello, world");
    }
}
```

Methods (cont.)

- General syntax for the `main` method:

```
public static void main(String[] args) {
     <statement>;
     <statement>;
     …
     <statement>;
}
```

  where each <statement> is replaced by a single instruction.

- Notes:
  - the main method always begins with the same *header*:
    `public static void main(String[] args)`
  - the code inside the method is enclosed in curly braces
  - each statement typically ends with a semi-colon
  - the statements are executed sequentially

Identifiers

- Used to name the components of a Java program like classes and methods.

- Rules:
  - must begin with a letter (a-z, A-Z), $, or _
  - can be followed by any number of letters, numbers, $, or _
  - spaces are not allowed
  - cannot be the same as a *keyword* – a word like `class` that is part of the language itself (see the book)

- Which of these are *not* valid identifiers?

`n1`	`num_values`	`2n`
`avgSalary`	`course name`	

- Java is *case-sensitive* (for both identifiers and keywords).
  - example: `HelloWorld` is not the same as `helloWorld`

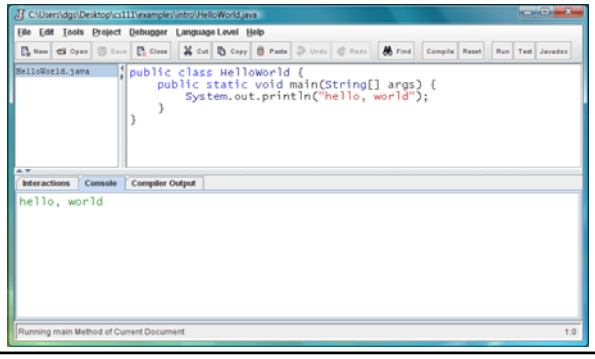
Conventions for Identifiers

- Capitalize class names.
  - example: `HelloWorld`
- Do not capitalize method names.
  - example: `main`
- Capitalize internal words within the name.
  - example: `HelloWorld`

Printing Text

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("hello, world");
    }
}
```

- Our program contains a single statement that prints some text.
- The printed text appears in a window known as the *console*.



Printing Text (cont.)

- The general format of such statements is:

```
System.out.println("<text>");
```

  where <text> is replaced by the text you want to print.

- A piece of text like "`Hello, world`" is referred to as a *string literal.*
  - string: a collection of characters
  - literal: specified explicitly in the program ("hard-coded")
- A string literal must be enclosed in double quotes.
- You can print a blank line by omitting the string literal:

```
System.out.println();
```

Printing Text (cont.)

- A string literal cannot span multiple lines.
  - example: this is *not* allowed:

```
System.out.println("I want to print a string
  on two lines.");
```

- Instead, we can use two different statements:

```
System.out.println("I want to print a string");
System.out.println("on two lines.");
```

println vs. print

- After printing a value, `System.out.println` "moves down" to the next line on the screen.
- If we don't want to do this, we can use `System.out.print` instead:

```
System.out.print("<text>");
```

The next text to be printed will begin *just after* this text – on the same line.

- For example:

```
System.out.print("I ");
System.out.print("program ");
System.out.println("with class!");
```

is equivalent to

```
System.out.println("I program with class!");
```

Escape Sequences

- Problem: what if we want to print a string that includes double quotes?
  - example: `System.out.println("Jim said, "hi!"");`
  - this won't compile. why?
- Solution: precede the double quote character by a \

```
System.out.println("Jim said, \"hi!\"");
```

- \" is an example of an *escape sequence*.
- The \ tells the compiler to interpret the following character differently than it ordinarily would.
- Other examples:
  - `\n` a newline character (goes to the next line)
  - `\t` a tab
  - `\\` a backslash