How to Run a Java Program

CSE 1310 – Introduction to Computers and Programming
Alexandra Stefan
University of Texas at Arlington

Outline

- Software installation jGRASP and Java bundle

- Create a Java program in jGRASP
- Run the program

- Error messages and the 3 types of errors

- Creating a program with existing code (create new and copy/paste)
- Comments
- Settings in jGRASP
- Program output with System.out.print() anb System.out.println()

Initial Steps

- In order to use the same environment as I will in class, you need to install Java and jGRASP on your machine. You have 2 options for this:
  - **Recommended and easier: install the bundle of them together**
  - Alternative 1: Install them separate
  - Alternative 2: if you already have Java, you can install just jGRASP.

- These are steps that you do just once. Once Java and jGRASP are installed, you can develop and run any Java program you like on your computer.

- Note:
  - Having Java on your machine is mandatory.
  - jGRASP is an IDE (Integrated Development Environment). Other IDEs such as Eclipse are also fine. We recommend jGRASP as it provides data visualization which helps you understand better how the program works.

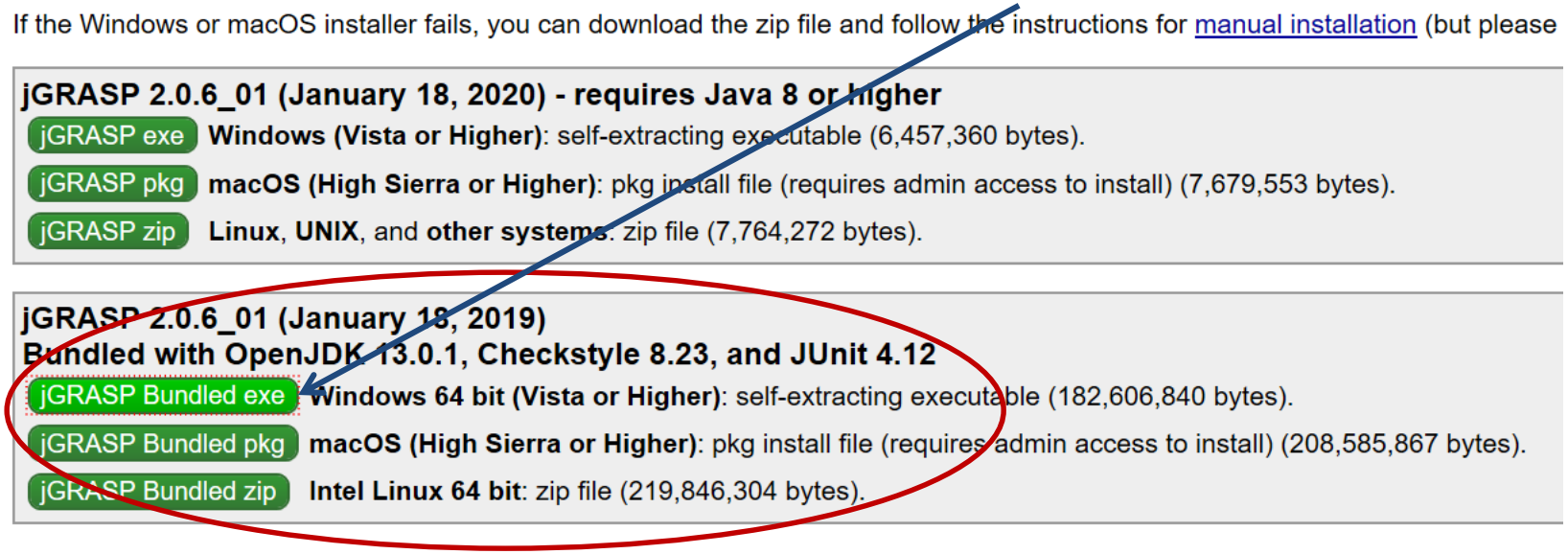
Installing the jGRASP and Java **bundle** for Windows 64 bit

From the jGRASP download page: https://spider.eng.auburn.edu/user-cgi/grasp/grasp.pl?;dl=download_jgrasp.html

Download the bundle for your operating system from:

jGRASP 2.0.6_01 (January 18, 2020) - requires Java 1.6 or higher
Bundled with OpenJDK 13.0.1, Checkstyle 8.23, and JUnit 4.12

E.g. for Windows 64 bit, download the jGrasp Bundled exe



Installing the jGRASP and Java **bundle** for Windows 64 bit

- Next double click the executable to start running it
- Use the default setting (accept license and click Next when prompted during installation)
- When the installation is complete, you will have a short cut on your desktop

- You can launch jGRASP by double clicking it. You can also starts it from the Windows program list.

- You finished the installation!

Note The installation for other systems should be similar. You can check the page from the University of Washington. They show the installation steps for installing Java and jGRASP separate (not as a bundle, like we did): https://courses.cs.washington.edu/courses/cse14x/software/jgrasp.html

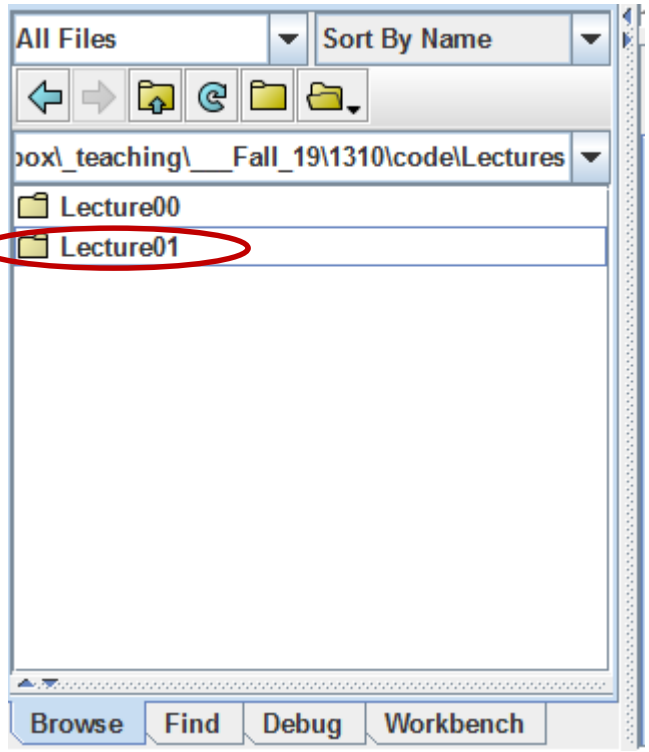
jGrasp or other IDE

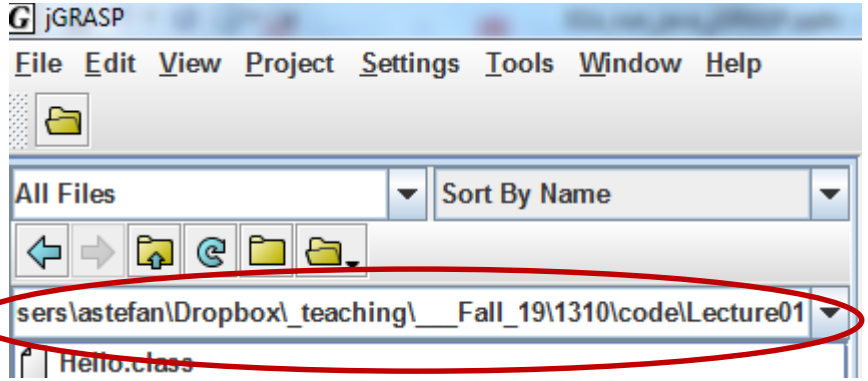
- jGRASP will be shown in the following slides, and will be used in class, but you can use other IDEs (e.g. Eclipse)

- See the “Getting Started” video tutorial from jGRASP: https://www.youtube.com/watch?v=DHICqIYV33k

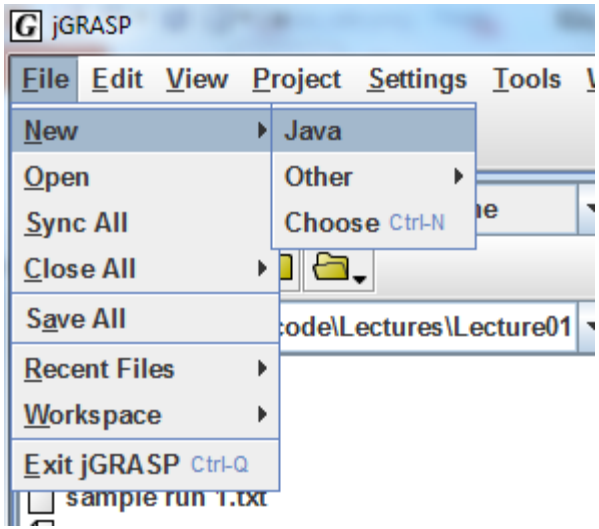
Using jGRASP

- Before you start using jGRASP, decide where you will store the Java code you write:
  - create a folder for this class on your machine. I will have a folder for lectures and one for homework. And inside there, a separate folder for each lecture or homework.
- When you start jGRASP, from the Browse tab navigate to the intended folder and the programs will be saved there.





Create a new Java file

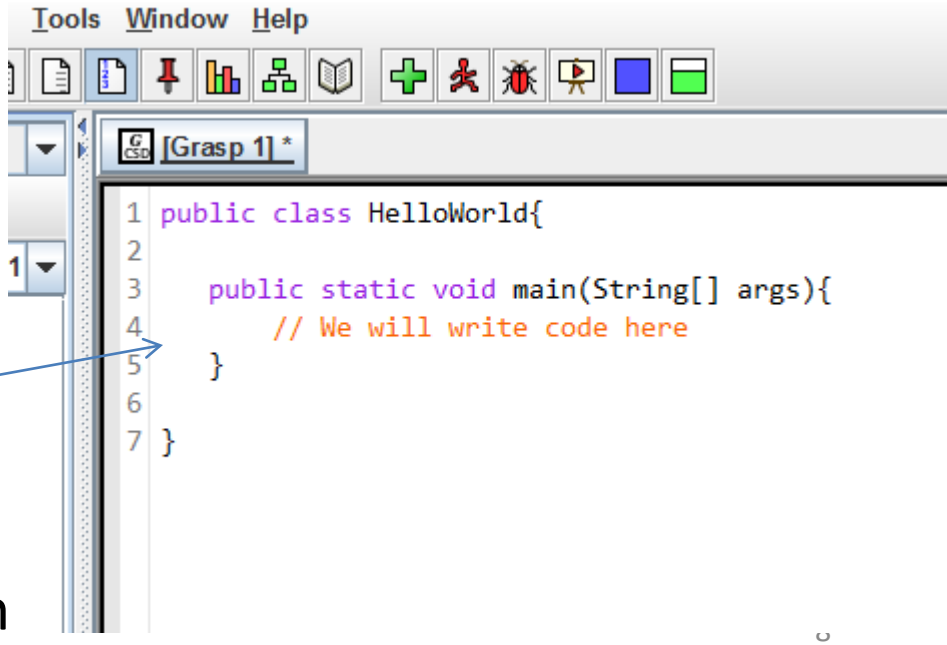


- To create an empty file:

  File -> Next-> Java

- Type this text in the window on the right:

```
public class HelloWorld{
   public static void main(String[] args){
      // We will write code here
   }
}
```

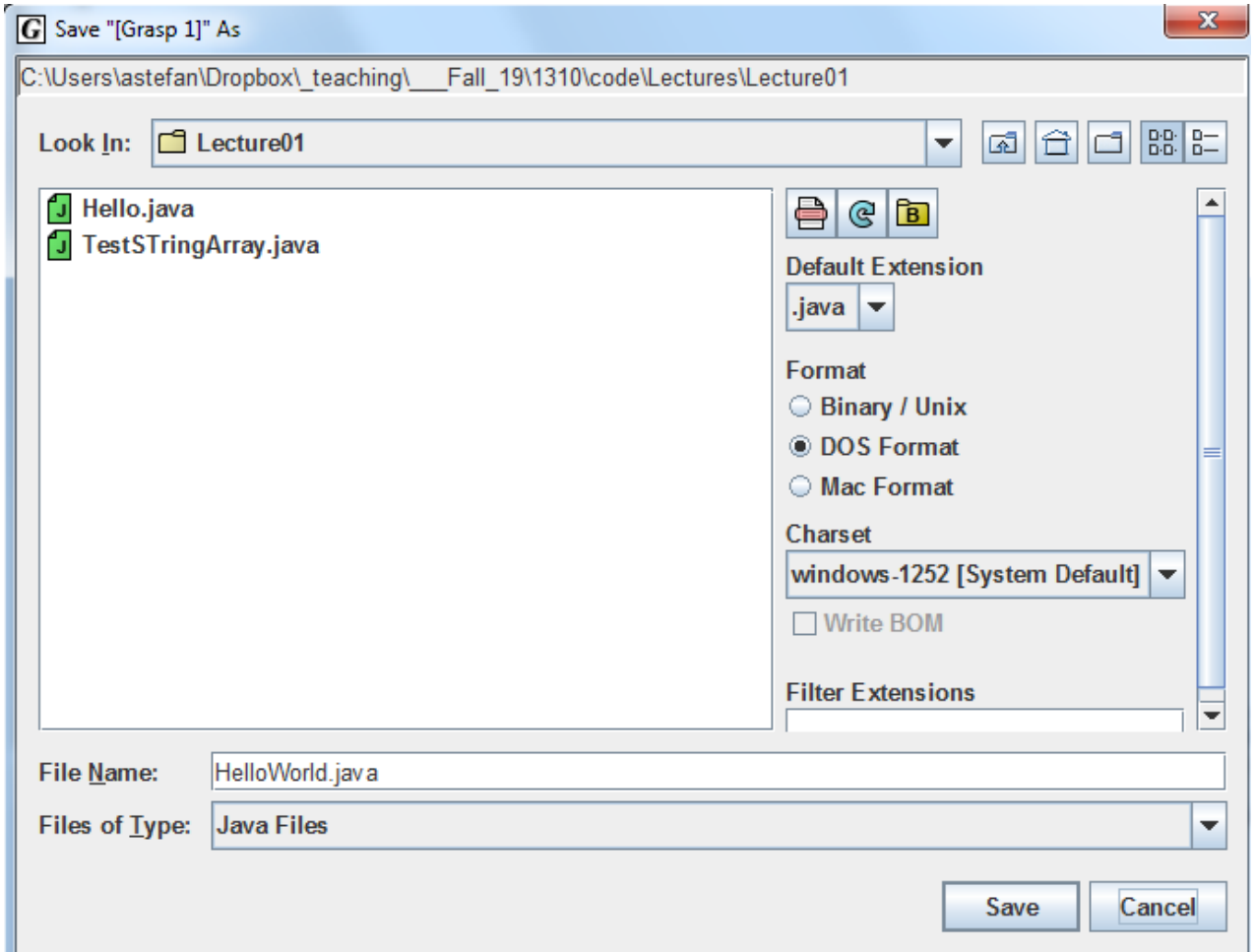


- It will look like this:
- ***HelloWorld*** is the class name. It will be different for every program

Create a new Java file continued

To save the file: *File->Save*

And the keep the suggested *File Name* (here *HelloWorld.java*).



Create a new Java file continued

- You can now :
  - See that file from File Explorer
  - In the jGRASP Browser tab
  - Compile the program with the compile button: 
  - Run the program with the run button: 
  - The .java file is what you will be submitting for your homework. NOT the .class file that is created after compiling.

Writing Code - comments

- The Java file that we created does not do anything yet.
- We can add code in the indicated area:

// We will write code here

Note that // indicates a “comment line” , meaning that Java will NOT to execute any of the text *following* it on the same line. E.g.:

```
System.out.println("Hello world"); // replace this line
```

Note: multiple line comments are indicated by start and end special symbols: /* */

E.g.:

```
/* this
is a comment
on multiple lines */
```

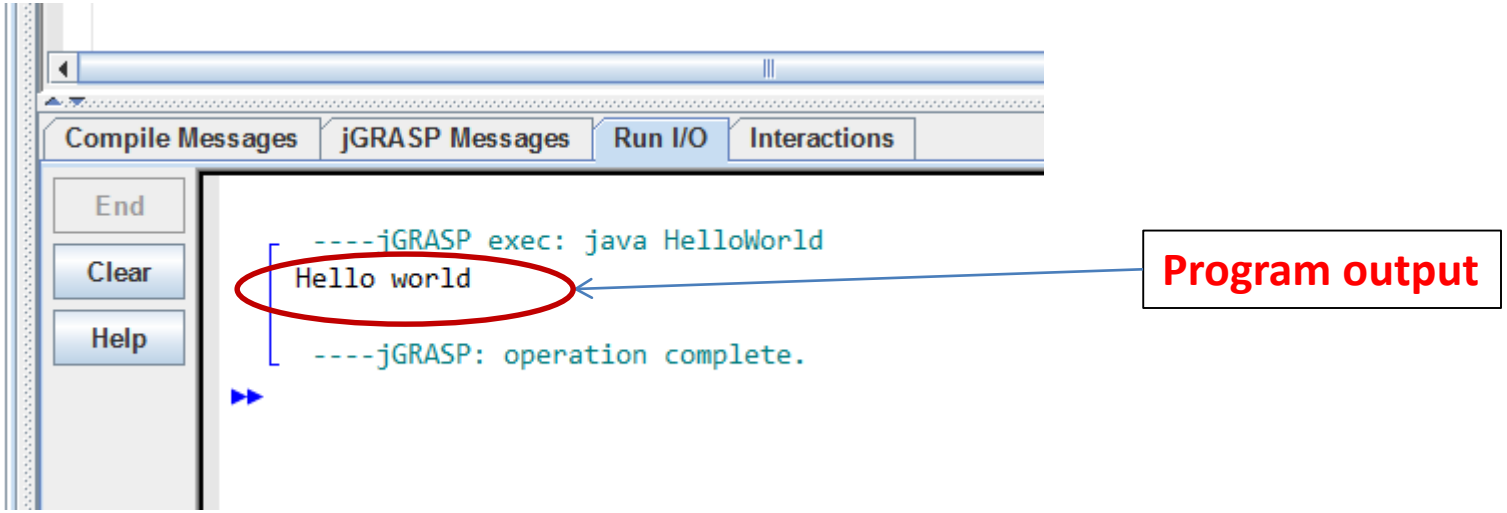
A First Example - Running the code

- Below we have replaced the placeholder with a line of code.

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

}
```

- To run this code, press the run button: 
- You will now see in the Run I/O tab the result of running the program. We will call this the program output.



- The Run I/O Window
  - is NOT the Source Code/Source File. It should not be submitted with the homework. It simply shows 'the result' of one execution of your program.

- See Figure 8 'From Source code to Running Program', page 9 in the textbook.

- If you close the window with the source code, you can open it by browsing to the folder where you put the .java file and clicking on the .java file.

Program syntax

Program starts executing the first instruction in main() method.
It executes instructions in given order.

Class name.
Must match the file name. (E.g. *HelloWorld*.java)
Use a name relevant to what the program does.

```
public class HelloWorld {

   public static void main(String[] args) {
      System.out.println("Hello world");
      System.out.println("Hello world again!");
   }

}
```

Keywords (reserved words)

Use indentation to make the code more readable

Match curly braces both for the class and for the main method.

Statements (typical lines of code) end with a semicolon ;
Exceptions : class and method declarations, complex instructions.

You must use the exact spelling for keywords, methods, and more to see.

Explore

- Try to violate syntax rules and see what happens:
  - Missing semicolon,
  - Move statements outside main()
  - Mismatch class name and filename
  - Modify spelling for keywords or main.

Template for your program:

- Start every program with this template.
- You can only modify the components in ***italic bold text***

```
public class ChoseYourClassName{

    public static void main(String[] args){
        // Replace this line with your code
    }

}
```

Failure to Run – Errors

- A lot of times you will get errors, and you will not be able to run the program.
- Do not panic, this is a very common thing to happen.
- For example, on this code we introduce an error on purpose, to see what happens (we delete the semicolon after "Hello world").

```
public class hello1 {
  public static void main(String[] args) {
    System.out.println("Hello world")
  }

}
```

- Now, when we try to run this code, we get what is shown on the next slide:

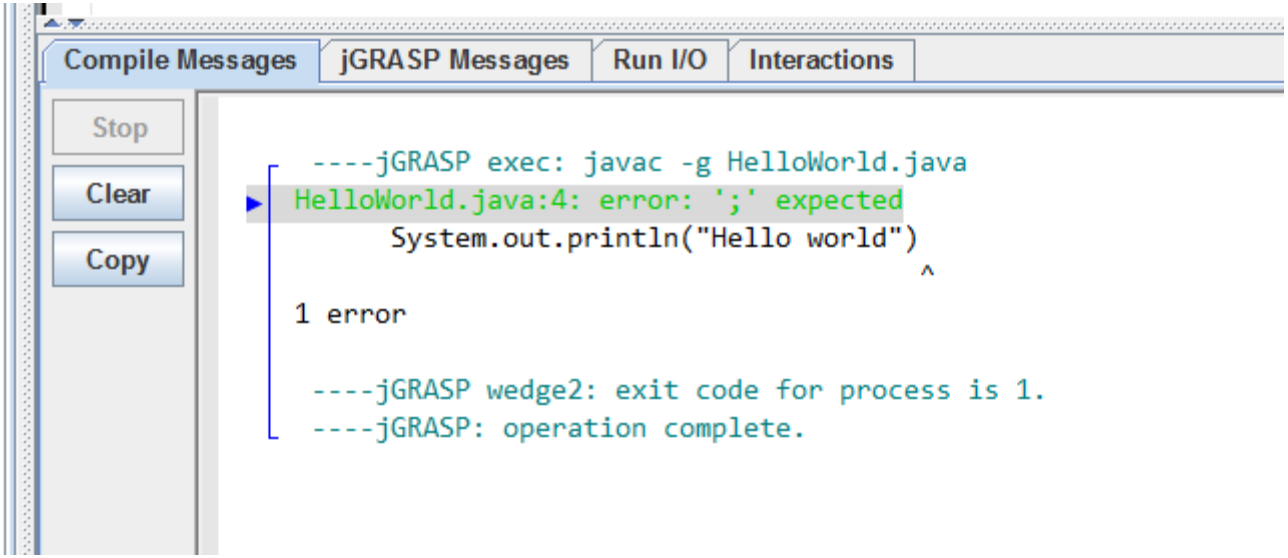
Error messages for Syntax errors

- The error message:

HelloWorld.java:4: error: ';' expected

- indicates the problem **';' expected**
- And the number **4** tells you what line of code has that problem

- You will see many error messages. You should not be scared by them, but instead use them to understand what is wrong with your program.



Errors

1. *Syntax errors*
   - You do not follow the rules for writing code
   - The compiler catches them and
   - Reports an error message
   - You cannot run the code until you fix it
2. *Logical errors (at runtime)*
   - the program runs, but does the wrong thing (e.g. incorrect computations). E.g. prints: Hello *Wold*
3. *Program crashes (at runtime)*
   - Program “crashes/fails” during execution.
   - E.g.: division by 0: `System.out.println(1/0);`
   - Give input of the wrong type (e.g. text instead of number)

Settings in jGRASP

- Show line numbers

Running Existing Code

- Oftentimes you may want to run code from somewhere (the slides, the textbook, the course website, etc.).
- To do that, create a new .java file, copy/paste the code in that file, save it and make sure you do not change the recommended FileName as the name of the file must match the name of the class. In our previous example the class was **HelloWorld** and the file was called **HelloWorld.java** and

Writing and running a program

- Writing vs running a program
  - Write a recipe vs cooking according to a recipe
  - Installing a program/game/app vs lunching that game

- Program output
  - what your program “sends out” while it runs
  - Most common form: It will “print to the screen”

- Program input
  - What data/information is sent into the program while it runs.
  - Most common form: from the keyboard

Program output

Program:

```
public class Hello1 {
  public static void main(String[] args) {
    System.out.println("Have a nice day.");
    System.out.println(6.3 + 12/7);
  }
}
```

Output (as is in jGRASP):

```
----jGRASP exec: java Hello1
Have a nice day.
7.3
----jGRASP: operation complete
```

Output (as we will show in the future):

```
Have a nice day.
7.3
```

We will call this **"program output"**.

From now on, we will only show the program output, without the jGRASP lines.

System.out.println()

```
public class Hello1 {
  public static void main(String[] args) {
    System.out.println("Have a nice day.");
    System.out.println(6.3 + 12/7);
  }
}
```

- Is a ***method*** (like a function in math)
- Takes only one argument
- Syntax: *System.out.println(**argument**);*
  - required & case sensitive: *System.out.println () ;*
  - Spaces: ok around *();* not ok within *System.out.println*
- If the argument is text (also called a ***string***), it must be enclosed in double quotes.
- If the argument is a numerical expression (e.g. **`6.3 + 12/7`**), then System.out.println calculates and prints the ***result*** of that expression.

Syntax of System.out.println

- Is each of these lines ***syntactically*** correct or not? If correct, what will it print?

```
System.out.println("6.3 + 12/7");
```

Correct, prints 6.3 + 12/7

Note that the argument here is ***text***.

```
System.out.println(6.3 + 12/7);
```

Correct, prints 7.3

Note that the argument here is a ***numerical expression***.

Syntax of System.out.println

- What is wrong with these lines?

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

System.out.println("hello")

System.out.println "hello";

System.out.println 6.3 + 12/7;

System.out.println "hello" ();

System.out.println 6.3 + 12/7 ();

System.Out.println("hello");

System.out.prntln("hello");
```

Syntax of System.out.println

- What is wrong with these lines?

```
System.out.println(hello);
```
Will not run. Missing quotes

```
System.out.println("hello")
```
Missing semicolon

```
System.out.println "hello";
```
Missing parentheses.

```
System.out.println 6.3 + 12/7;
```
Missing parentheses.

```
System.out.println "hello" ();
```
Misplaced parentheses.

```
System.out.println 6.3 + 12/7 ();
```
Misplaced parentheses.

```
System.Out.println("hello");
```
Case sensitive: **o**ut

```
System.out.prntln("hello");
```
Typo: prntln

Printing " and \

- In order to print (as output) the quotation symbol ", use \"

```
System.out.println("She said \"Bye-bye!\" and left.");
```

- In order to print \ as output, use \\:

```
System.out.println("C:\\Users\\alex");
```

Exact Syntax

- If you do not follow the syntax **EXACTLY**, Java will refuse to execute that line.
- This is true not only for System.out.println, but for all the instructions and other syntax rules that we will see in this course.

System.out.print()

- Same as System.out.println() but does NOT move to a new line after printing

- Use the print and println method to write different programs that produce the same output.
  - E.g. Print Hello World! Using 2 (or 3) System.out.print() statements.

Review and terms

- Program syntax
  - Matching {}
  - ; at end of statement
  - Class name must match file name
- Compile vs run
- Source code vs program output
- The 3 types of errors
- Keywords: public, static, void, class
- Syntax and syntax rules
- Printing methods and their syntax:
  - System.out.print(…);
  - System.out.println(…);
- Program execution
  - Where it starts (first statement in *main()*)
  - Executes line by line in given order
- Comments:
  - One line
  - Multiple lines