CENG 112
NESNEYE YÖNELİK PROGRAMLAMA

Spring 2025
Gürhan Gündüz

Office hours: Pazartesi 13:30-15:00

email: ggunduz@pau.edu.tr

Goal of the Course

- Object Oriented Programming Concepts

- Practice Object Oriented Programming using Java

- Be able to develop small scale applications using Java

Course Overview

- **Week 1** - Introduction & Basic Elements of Programming, Basic Elements of Programming
- **Week 2** – Loops, arrays
- **Week 3** - Methods, Strings
- **Week 4** - Objects & Classes
- **Week 5** – Objects & Classes, Exception Handling
- **Week 6** - Inheritance & Polymorphism

Course Overview

- **Week 7** - Inheritance & Polymorphism
- **Week 8** - Midterm Exam(it can change)
- **Week 9** - Interface and Abstract Classes
- **Week 10**- Collections, Generics
- **Week 11**- Database
- **Week 12-** Database
- **Week 13-** Java FX **?**
- **Week 14-** Java FX ?

Grading

- **In accordance with University policy, all students must be present for 70% of classroom instruction.**

• **Homework**	**10 %**
• **Midterm**	**40 %**
• **Final exam**	**50 %**

Logistics

- Textbook:
  - Java: A Beginner's Guide, Seventh Edition 7th Edition by Herbert Schildt
  - Head First Java, 2nd Edition by Bert Bates, Kathy Sierra
  - Deitel, Java How to Program, 11/e, Early Objects
  - Any other resources you can find

Logistics

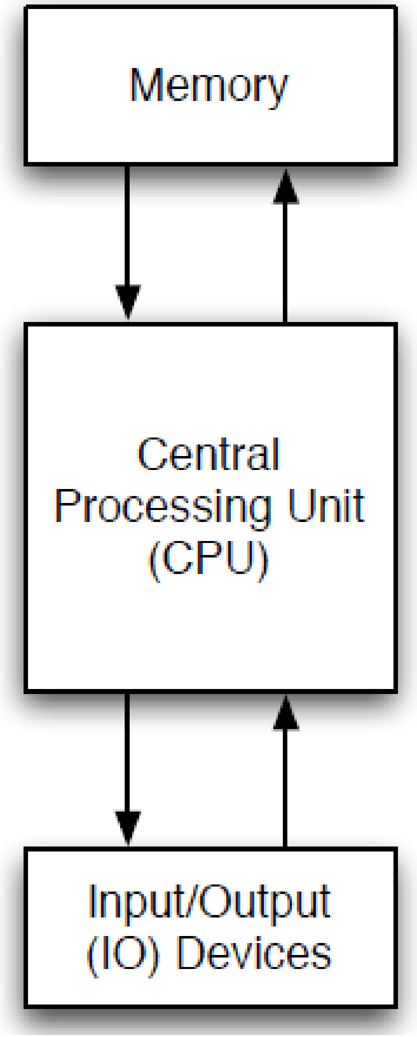
- BilMoodle or EDS will be used for homework submission
  - https://bilmoodle.pau.edu.tr/
  - EDS can be accessed over “Pusula Bilgi Sistemi”

Lab Grading

- You should commit the work you have completed to BilMoodle/EDS
- To get a grade
  - Make sure it is compilable
  - It gives the correct output
  - Its implementation is effective and efficient

The Computer

The Computer



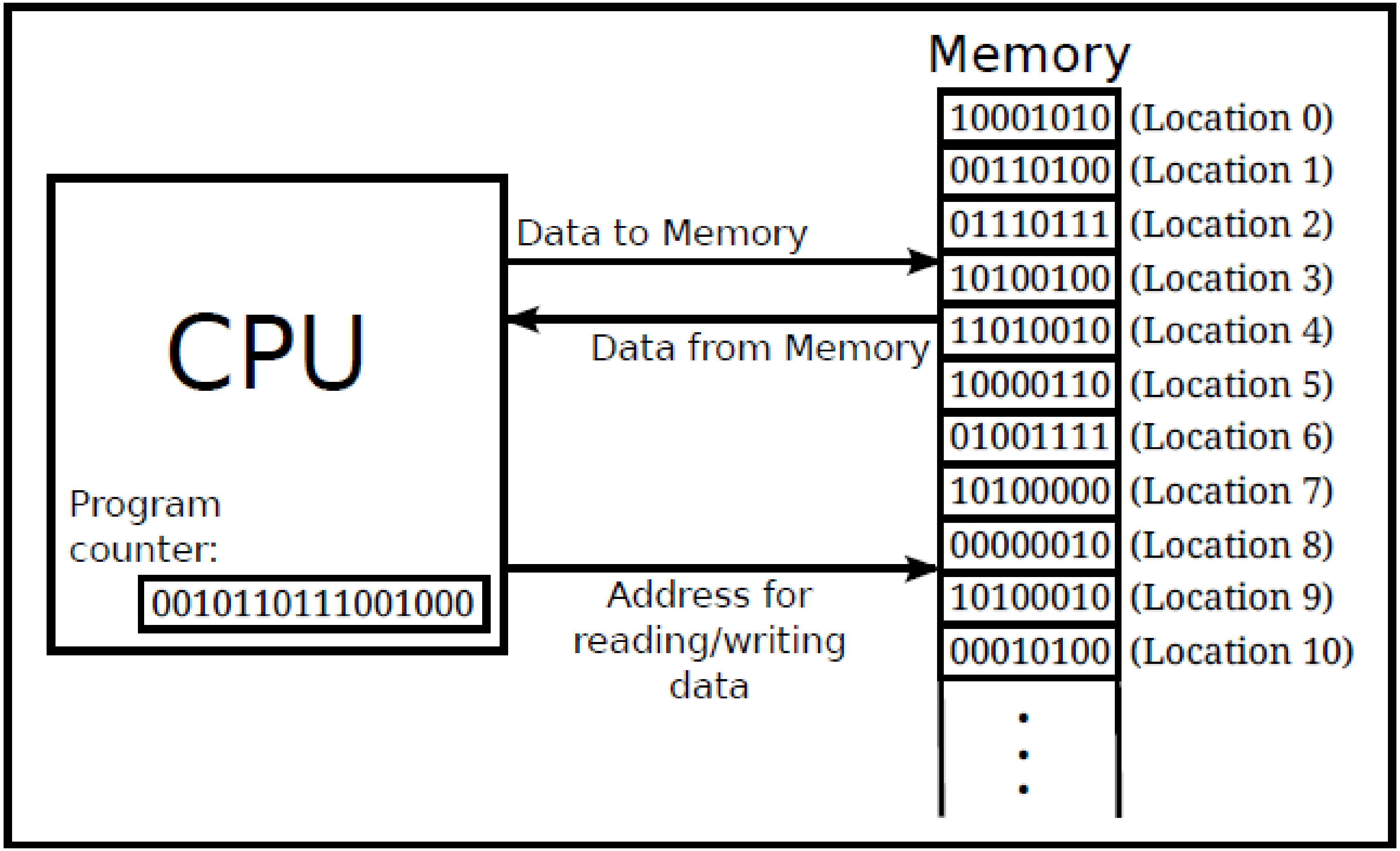
The Computer

- CPU (Central Processing Unit)
  - execute programs

- Computer Program
  - a sequence of instructions that can be processed mechanically by a computer

- Machine Language
  - lowest-level representation of computer programs that can be executed by the computer

How Program is Executed



10001010	(Location 0)
00110100	(Location 1)
01110111	(Location 2)
10100100	(Location 3)
11010010	(Location 4)
10000110	(Location 5)
01001111	(Location 6)
10100000	(Location 7)
00000010	(Location 8)
10100010	(Location 9)
00010100	(Location 10)

How Program is Executed

- Main memory holds
  - – machine language programs and
  - – data

- The CPU fetches
  - – machine language instructions from memory one after another and executes them

CPU Instructions

z = x + y

Read location x

Read location y

Add

Write to location z

Programming Languages

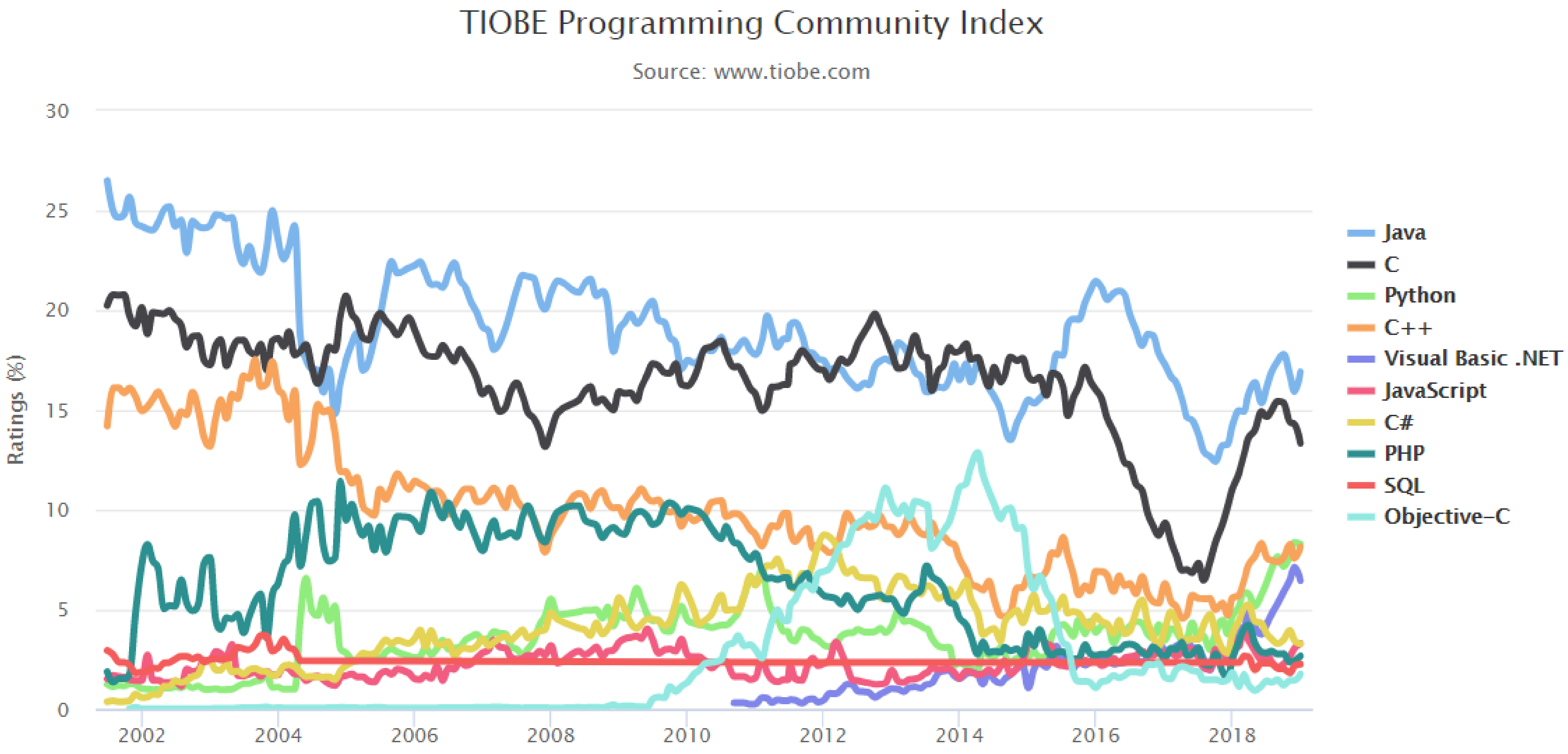
- Easier to understand than CPU instructions
- Needs to be translated for the CPU to understand it

Java

Java

- “Most popular” language
- Runs on a “virtual machine” (JVM)
- More complex than some (eg. Python)
- Simpler than others (eg. C++)

Java is popular



Why Java?

- Object Oriented Programming Language
- Portable
  - offers a write-once-run-anywhere with the help of virtual machine
- Backward compatibility
  - Old programs survive while the language evolves
- Scalability and Performance
  - is used in large enterprise applications and big data projects

Why Java?

- Huge Open Source Community and Many Libraries
  - http://apache.org/

- Various Nice Integrated Development Environments
  - NetBeans(We are going to use this)
  - Eclipse

Programming Environment

- Java “Standard Edition”
  - Java Runtime Environment (JRE)
    - does not allow you to compile your Java sources
  - Java Development Kit (JDK)
    - You need to install JDK for use in this course
- There are two alternatives
  - Command line environment and a Text Editor
  - Integrated Development Environment (IDE)

Obtaining the Java Development Kit

- The JDK can be downloaded from
  - https://www.oracle.com/technetwork/java/javase/downloads/index.html

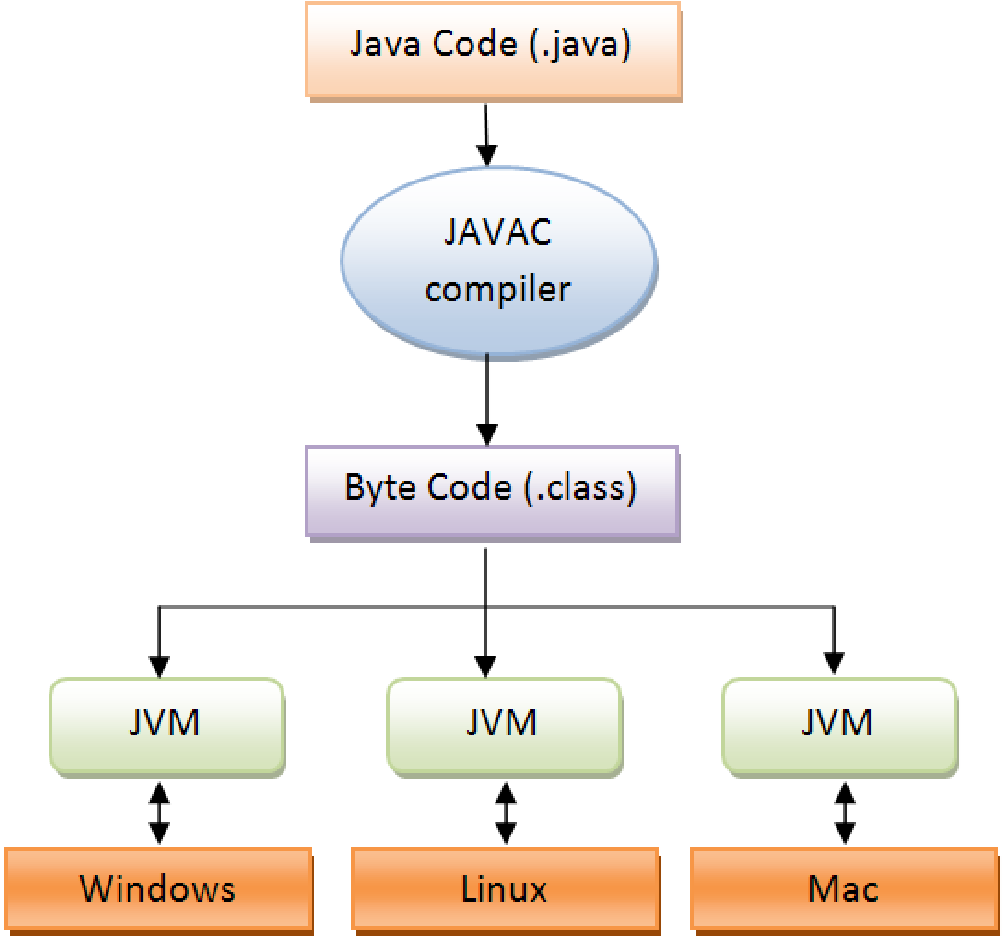
- Go to the download page and follow the instructions for the type of computer that you have

Checking Installed JDK

- Type the following commands
  - java -version
  - javac -version

- If you get a message such as “Command not found,” then there is a problem in your installation

Java Virtual Machine (JVM)


Java Code (.java)

JAVAC compiler

Byte Code (.class)

JVM

JVM

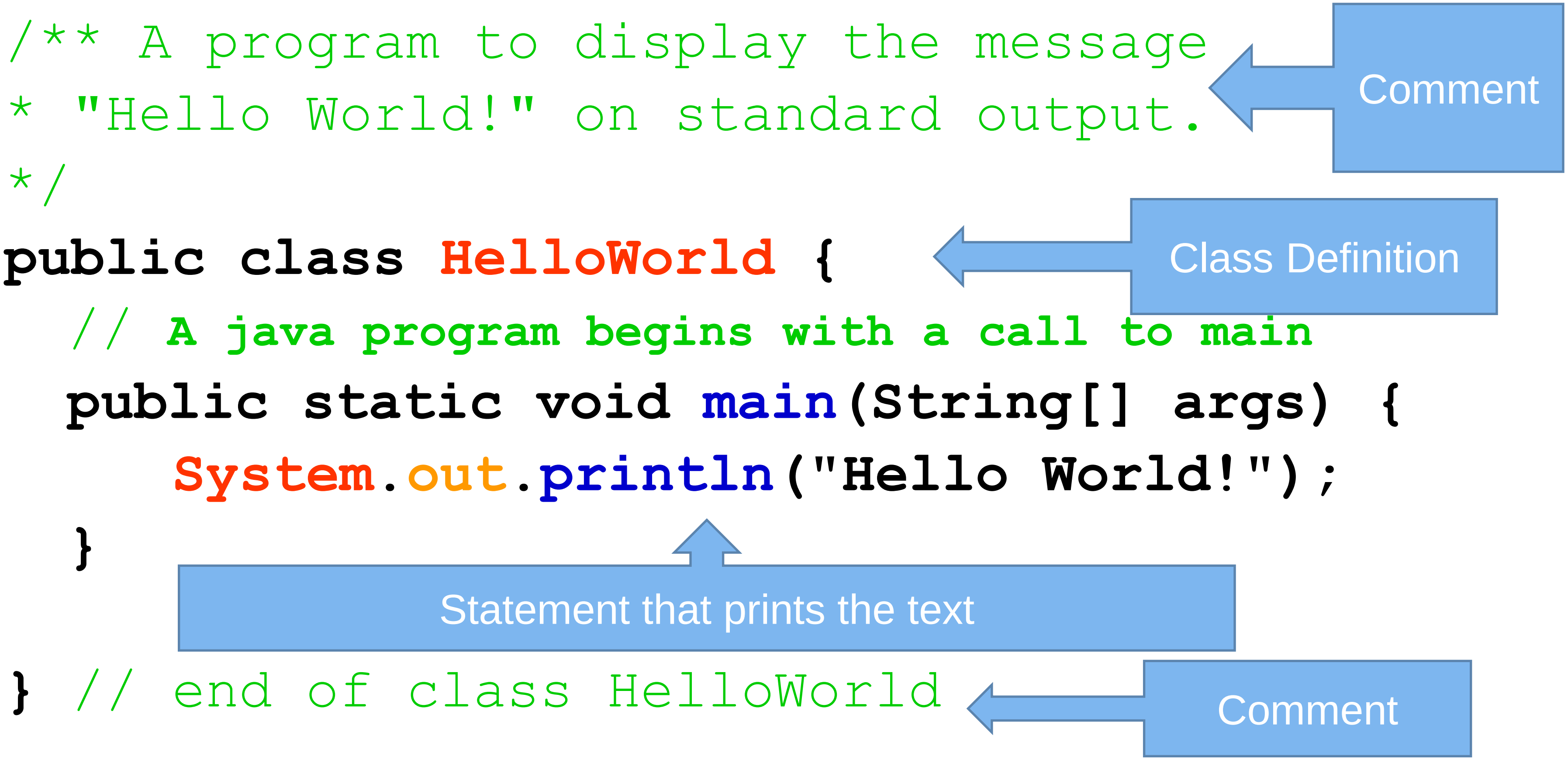
JVM

Windows

Linux

Mac

HelloWorld.java



```
/** A program to display the message
* "Hello World!" on standard output.
*/
public class HelloWorld {
  // A java program begins with a call to main
  public static void main(String[] args) {
     System.out.println("Hello World!");
  }
} // end of class HelloWorld
```

Comment

Class Definition

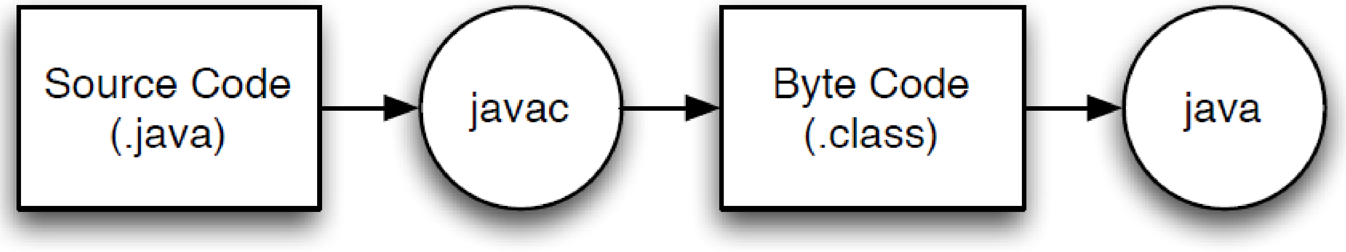
Statement that prints the text

Comment

Development

- Three steps
  - Write your code to a file with java extension (such as HelloWorld.java)
  - Compile your file using “javac” command
    - javac HelloWorld.java
  - Run your binary using “java” command
    - java HelloWorld

Compiling Java

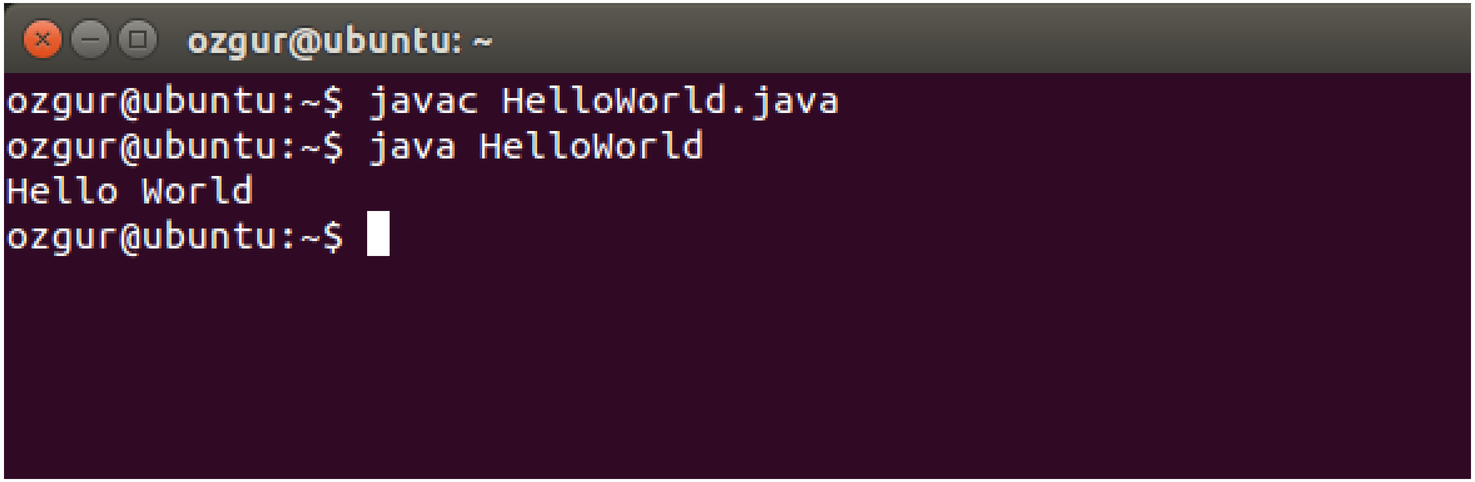


Compiling and Running

- javac HelloWorld.java
  - this command will produce a file "HelloWorld.class" unless you do not have an error in the source file

- java HelloWorld
  - This command will execute "HelloWorld.class"
  - Note that the extension (.class) is not specified in the command

HelloWorld.java



```
ozgur@ubuntu:~$ javac HelloWorld.java
ozgur@ubuntu:~$ java HelloWorld
Hello World
ozgur@ubuntu:~$
```

Java vs Python

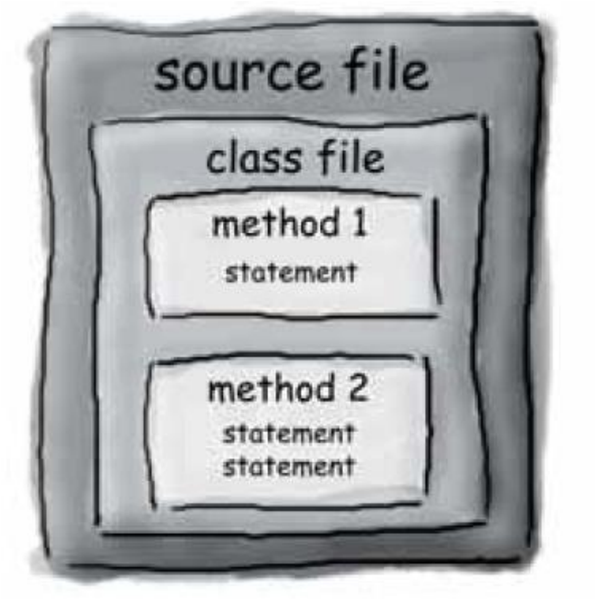
- Python is an interpreted language.
  - Execute the code directly and freely, without previously compiling a program into machine-language instructions.
- Java is a compiled language
  - Translates the source code through a compiler. This results in very efficient code that can be executed any number of times.

Program Structure

```
public class CLASSNAME {

  public static void main(String[] arguments){
      STATEMENTS
  }

}
```

Program Structure

```
public class Dog{
        int var;
        void bark(){
                //statements
        }

        void eat(){
                //statements
        }

}
```



Put a class in a source file.

Put methods in a class.

Put statements in a method.

Java has complex syntax

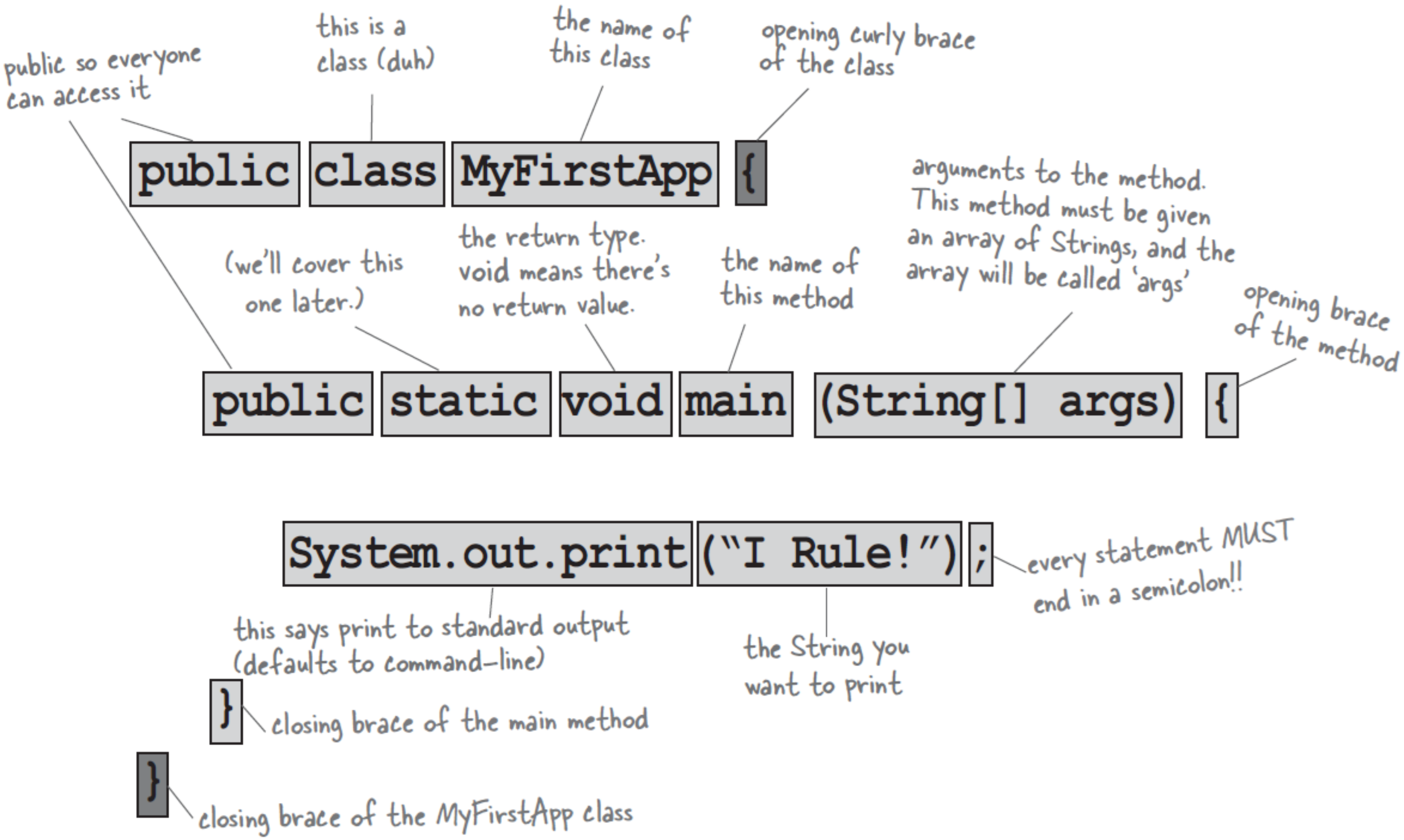
- **JAVA**

```
public class HelloWorld {

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

}
```

- **PYTHON**

```
print("Hello, World!")
```

Anatomy of Class



```
public class MyFirstApp {
public static void main (String[] args) {
System.out.print("I Rule!");
}
}
```

public so everyone can access it

this is a class (duh)

the name of this class

opening curly brace of the class

(we'll cover this one later.)

the return type. void means there's no return value.

the name of this method

arguments to the method. This method must be given an array of Strings, and the array will be called 'args'

opening brace of the method

every statement MUST end in a semicolon!!

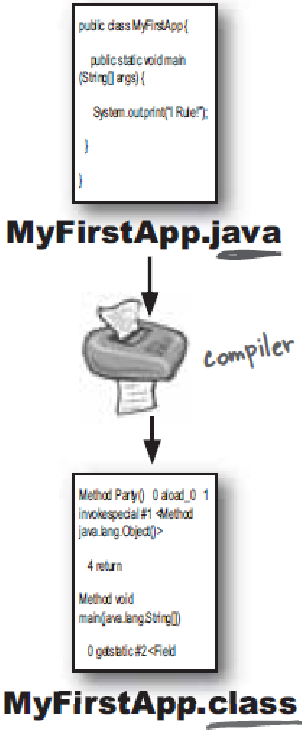
this says print to standard output (defaults to command-line)

the String you want to print

closing brace of the main method

closing brace of the MyFirstApp class

My First Application



```
public class MyFirstApp {

   public static void main
(String[] args) {

     System.out.print("I Rule!");

   }

}
```

```
Method Party()  0 aload_0  1
invokespecial #1 <Method
java.lang.Object()>

  4 return

Method void
main(java.lang.String[])

  0 getstatic #2 <Field
```

```
public class MyFirstApp {

   public static void main (String[] args) {
      System.out.println("I Rule!");
      System.out.println("The World");
   }

}
```

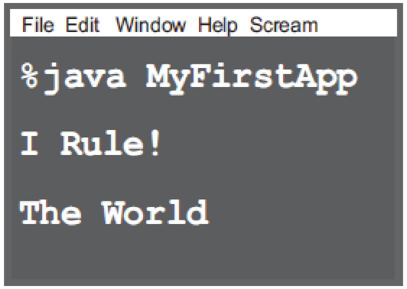
❶ Save

`MyFirstApp.java`

❷ Compile

`javac MyFirstApp.java`

❸ Run



```
%java MyFirstApp
I Rule!
The World
```

public static void main (String args[])

- Java applications begin execution by calling main( )
  - **public** keyword is an access modifier
  - The keyword **static** allows main( ) to be called before an object of the class has been created
  - **void** simply tells the compiler that main( ) does not return a value
  - there is only one parameter, **String args[]**, which declares a parameter named args.

Syntax Issues

- Semicolon “;” is required at the end of statements

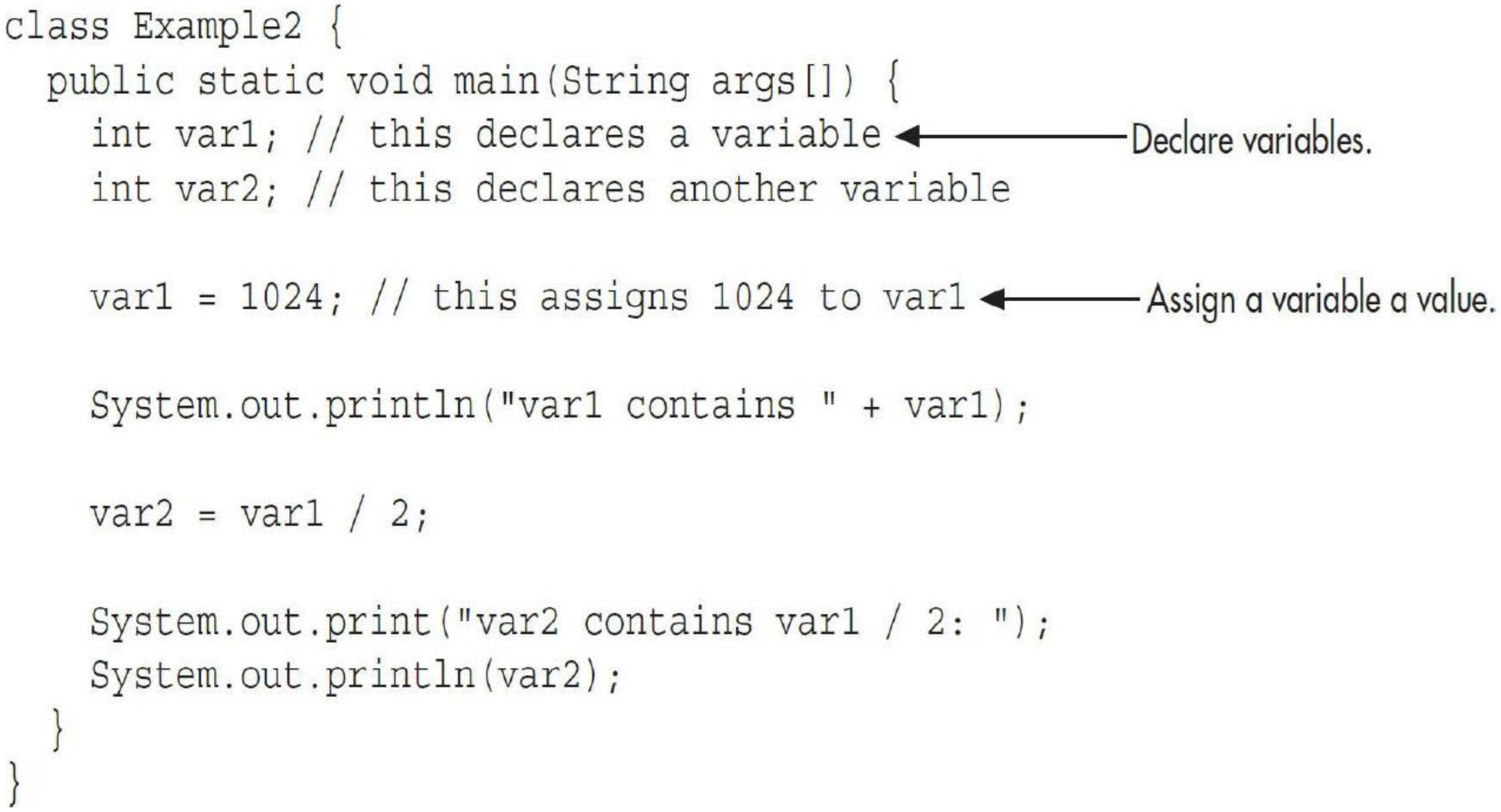
- Curly braces “{ }” are used to enclose code block such as method bodies, class content, etc.

- Improper usage will cause error during compilation

Variables

- A variable is a named memory location that can be assigned a value.
- The value of a variable can be changed during the execution of a program.
- The following program creates two variables called var1 and var2:

Variables



```
class Example2 {
  public static void main(String args[]) {
    int var1; // this declares a variable
    int var2; // this declares another variable

    var1 = 1024; // this assigns 1024 to var1

    System.out.println("var1 contains " + var1);

    var2 = var1 / 2;

    System.out.print("var2 contains var1 / 2: ");
    System.out.println(var2);
  }
}
```

Declare variables.

Assign a variable a value.

Variables

- All variables must be declared before they are used
- The **type** of values that the variable can hold must also be specified
  - var1 can hold integer values
  - to declare a variable to be of type integer, precede its name with the keyword **int**
- Declare a variable you will use a statement like this:
  - **type var-name;**

Types

- Kinds of values that can be stored and manipulated.
  - **boolean:** Truth value (true or false).
  - **int:** Integer (0, 1, -47).
  - **double:** Real number (3.14, 1.0, -2.1).
  - **String:** Text (“hello”, “example”).

Primitive Types

- There are eight so-called primitive types
  - **boolean:** Truth value (true or false).
  - **short:** corresponds to two bytes (16 bits). range -32768 to 32767.
  - **int:** corresponds to four bytes (32 bits). range -2147483648 to 2147483647.
  - **long** corresponds to eight bytes (64 bits). range -9223372036854775808 to 9223372036854775807.

Primitive Types

- There are eight so-called primitive types
  - The **float** and **double** types hold real numbers (such as 3.6 and -145.99). Again, the two real types are distinguished by their range and accuracy.
  - A variable of type **byte** holds an 8-bit integer, which can represent any of the integers between -128 and 127, inclusive.
  - A variable of type **char** holds a single character.

Basic Language Elements

Variables

- Named location that stores a value of one particular type.

  ```
  TYPE NAME;
  ```

- Examples:

  ```
  int age;
  String name;
  double salary;
  ```

Assignment

- Use "=" to give variables a value.

- Example:

```
String code;   //declares String typed code variable

code = “CENG 112”; //assigns value to code
```

Assignment

- Can be combined with a variable declaration.

- Example:

```
double pi = 3.14;
boolean isJanuary = false;
```

- Write a code that will swap the values of two variables

HelloWorldVar.java

```
/** A program to display the message
* "Hello World!" on standard output.
*/
public class HelloWorldVar {

  public static void main(String[] args) {
     String message = "Hello World!";
     System.out.println(message);
     message = "Hello Again!";
     System.out.println(message);
     Scanner scn=new Scanner(System.in);
     String str=scn.nextLine();
     System.out.println(str);
  }
} // end of class HelloWorldVar
```

Operators

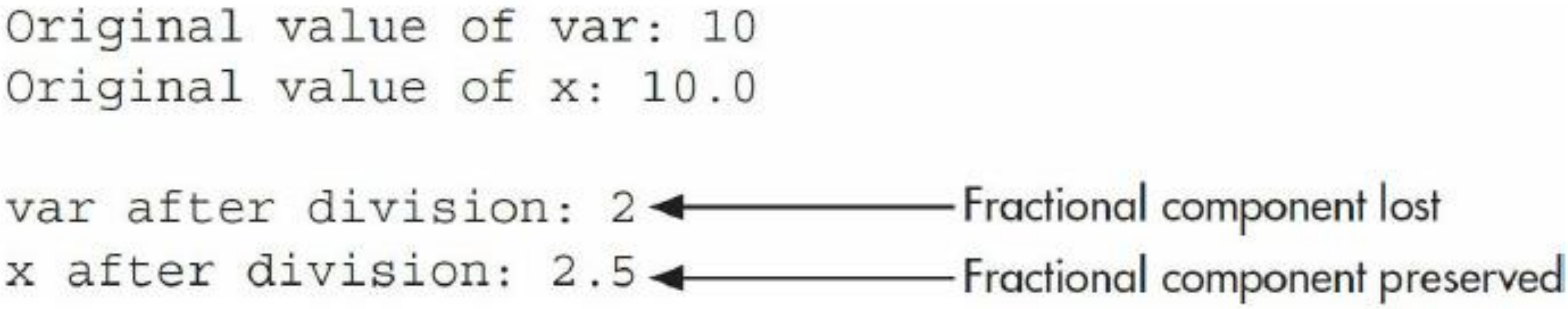
- Symbols that perform simple computations
  - Assignment: =
  - Addition: +
  - Subtraction: -
  - Multiplication: *
  - Division: /
  - Modulus: %

```
/*
   This program illustrates the differences
   between int and double.

   Call this file Example3.java.
*/
class Example3 {
  public static void main(String args[]) {
    int var; // this declares an int variable
    double x; // this declares a floating-point variable

    var = 10; // assign var the value 10

    x = 10.0; // assign x the value 10.0

    System.out.println("Original value of var: " + var);
    System.out.println("Original value of x: " + x);
    System.out.println(); // print a blank line

    // now, divide both by 4
    var = var / 4;
    x = x / 4;

    System.out.println("var after division: " + var);
    System.out.println("x after division: " + x);
  }
}
```

Output a blank line.

Example 3

- The output from this program is shown here:



```
Original value of var: 10
Original value of x: 10.0

var after division: 2
x after division: 2.5
```

Fractional component lost

Fractional component preserved

Division

- Division (“/”) operates differently on integers and on doubles!
  - double a = 5.0/2.0; // a = 2.5
  - int b = 4/2; // b = 2
  - int c = 5/2; // c = 2
  - double d = 5/2; // d = 2.0

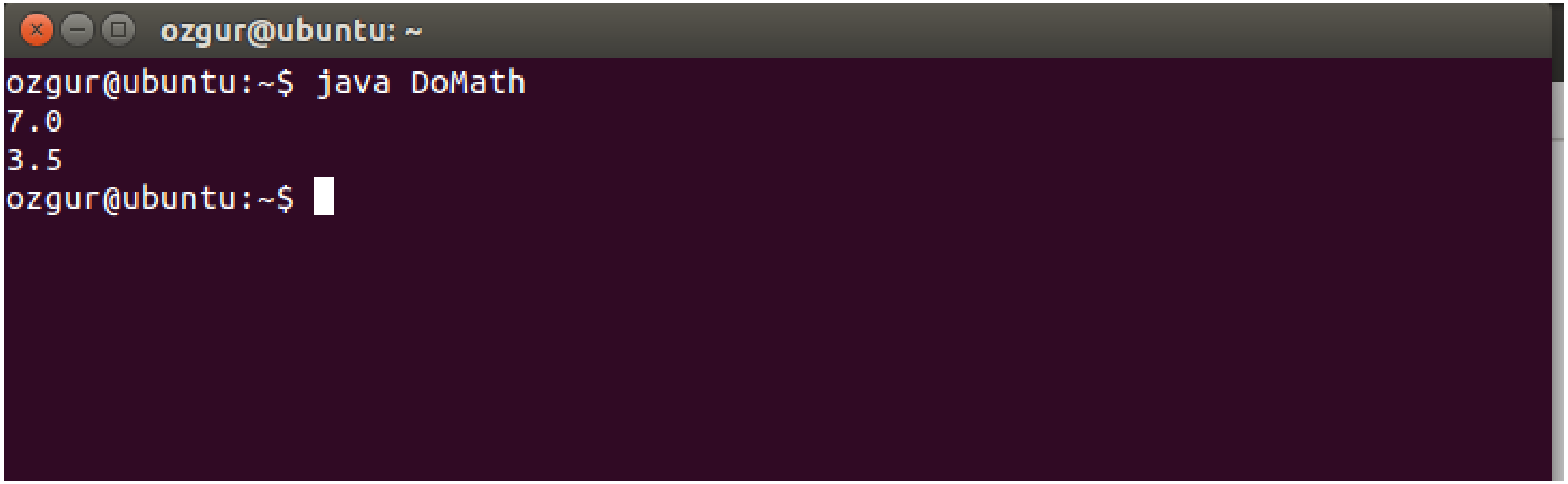
Order of Operations

- Follows standard math rules:
  1. Parentheses
  2. Multiplication and division
  3. Addition and subtraction

DoMath.java

```
public class DoMath {
  public static void main(String[] args){
        double score = 1.0 + 2.0 * 3.0;
        System.out.println(score);
        score = score / 2.0;
        System.out.println(score);
  }
}
```

DoMath.java



```
ozgur@ubuntu:~$ java DoMath
7.0
3.5
ozgur@ubuntu:~$ 
```

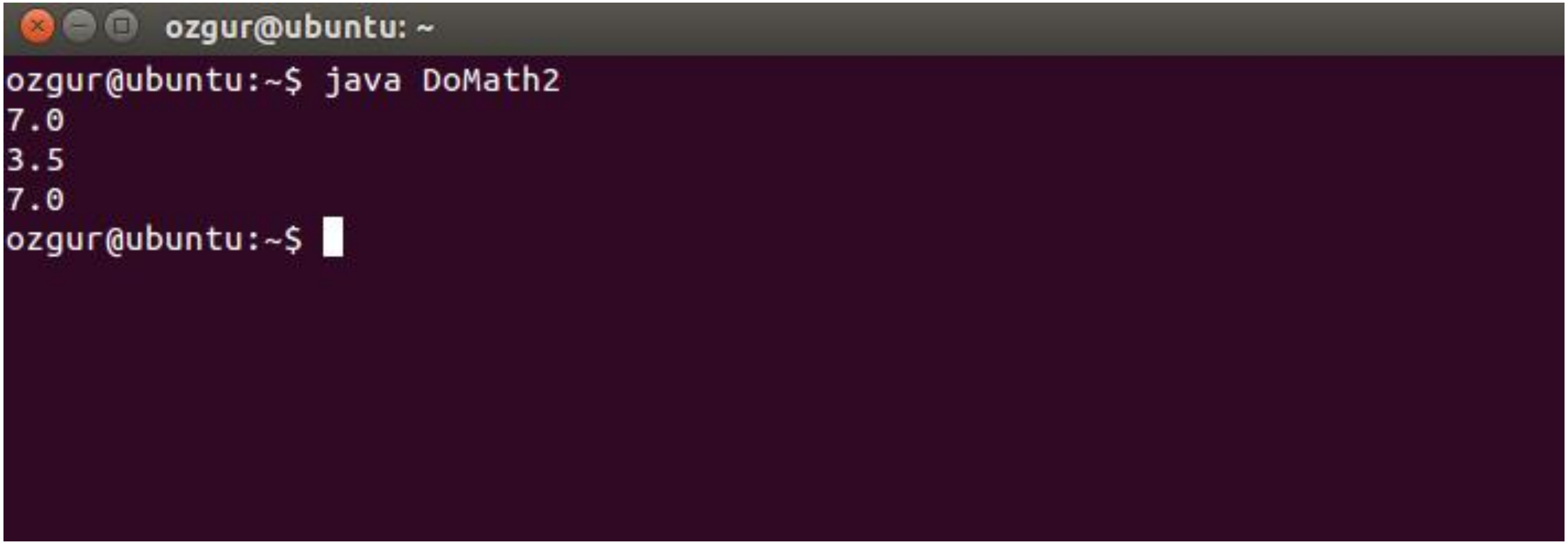
Assignment of Primitive Variables

- When you assign one primitive-type variable to another
  - The variable on the left receives a copy of the value of the variable on the right.

DoMath2.java

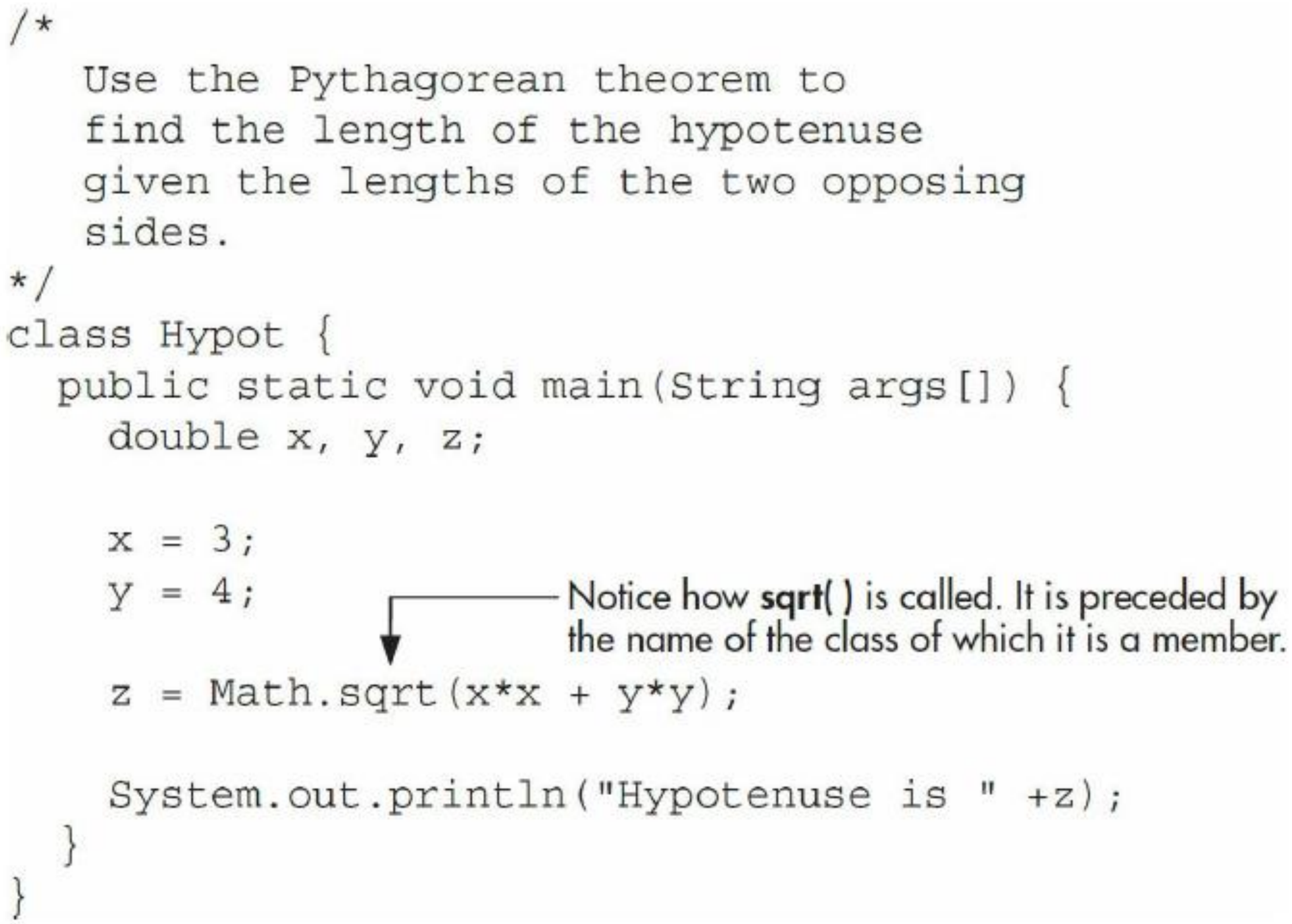
```
public class DoMath2 {
   public static void main(String[] args){
        double score = 1.0 + 2.0 * 3.0;
        System.out.println(score);
        double copy = score;
        copy = copy / 2.0;
        System.out.println(copy);
        System.out.println(score);
   }
}
```

DoMath2.java



```
ozgur@ubuntu:~$ java DoMath2
7.0
3.5
7.0
ozgur@ubuntu:~$ 
```

Hypotenuse Example



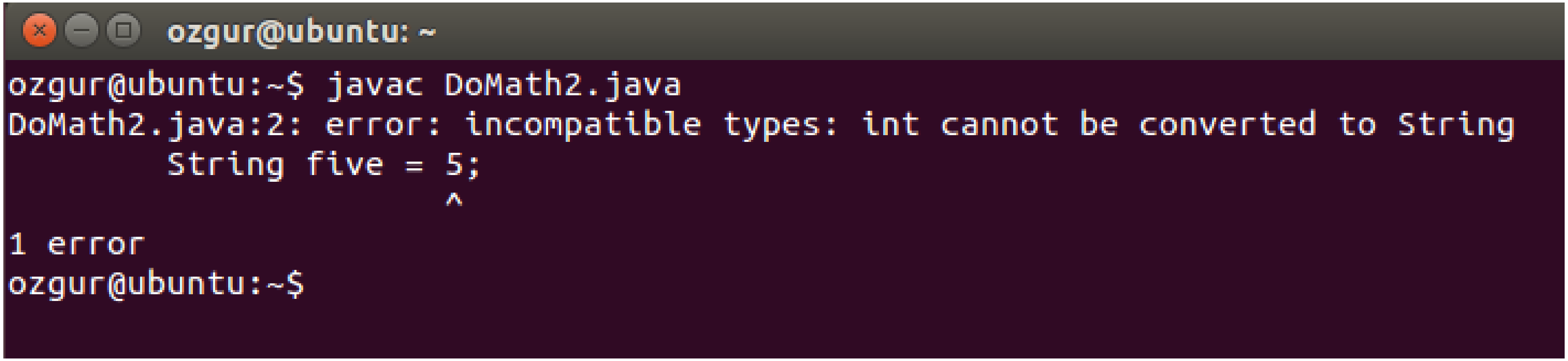
```
/*
   Use the Pythagorean theorem to
   find the length of the hypotenuse
   given the lengths of the two opposing
   sides.
*/
class Hypot {
  public static void main(String args[]) {
    double x, y, z;

    x = 3;
    y = 4;

    z = Math.sqrt(x*x + y*y);

    System.out.println("Hypotenuse is " +z);
  }
}
```

Notice how **sqrt( )** is called. It is preceded by the name of the class of which it is a member.

Mismatched Types

- Java verifies that types always match:

String five = 5; // ERROR!



```
ozgur@ubuntu:~$ javac DoMath2.java
DoMath2.java:2: error: incompatible types: int cannot be converted to String
        String five = 5;
                      ^
1 error
ozgur@ubuntu:~$
```

Conversion by casting

```
int a = 2;              // a = 2
double a = 2;           // a = 2.0 (Implicit)
int a = 18.7;           // ERROR
int a = (int)18.7;      // a = 18(explicit)
double a = 2/3;         // a = 0.0
double a = (double)2/3; // a = 0.6666…
```

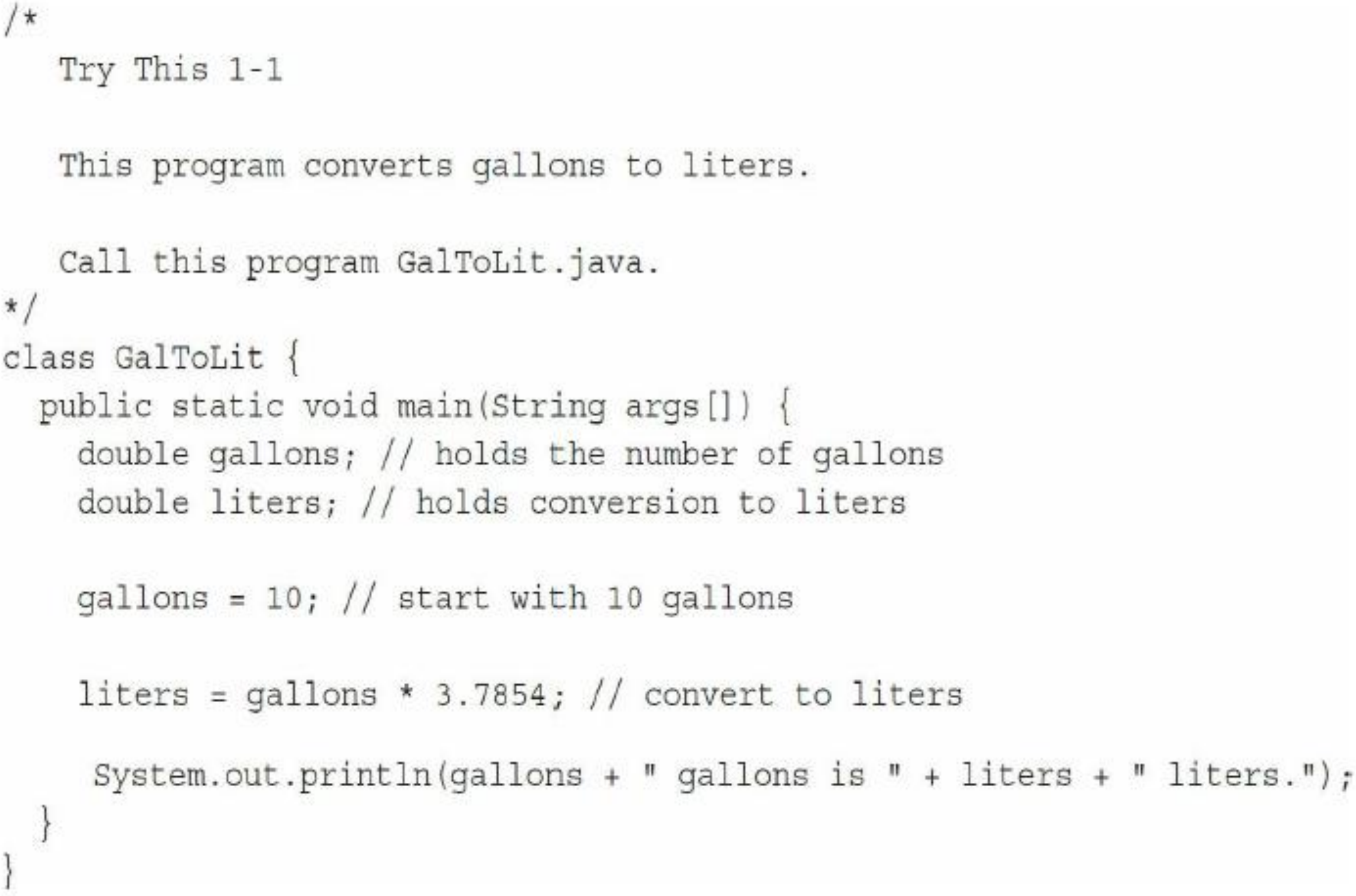
String Concatenation (+)

```
String text = "hello" + " world";
text = text + " number " + 5;
// text = "hello world number 5"

text = "hello" + 4 + 8;
//What will be the value of text?
```

Example: Gallon to Liter

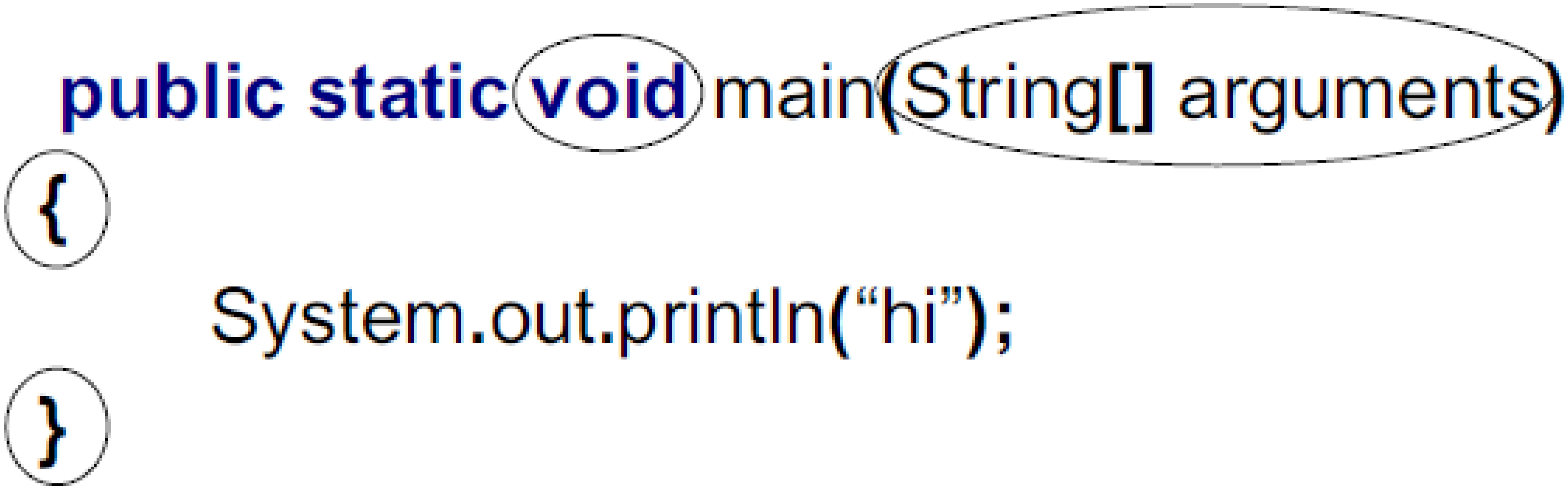
- Write a program that converts gallons to liters, given that there are 3.7854 liters in a gallon

Implementation



```
/*
   Try This 1-1

   This program converts gallons to liters.

   Call this program GalToLit.java.
*/
class GalToLit {
  public static void main(String args[]) {
    double gallons; // holds the number of gallons
    double liters; // holds conversion to liters

    gallons = 10; // start with 10 gallons

    liters = gallons * 3.7854; // convert to liters

     System.out.println(gallons + " gallons is " + liters + " liters.");
  }
}
```

Methods



```
  public static void main(String[] arguments)
{
        System.out.println("hi");
}
```

Control Statements

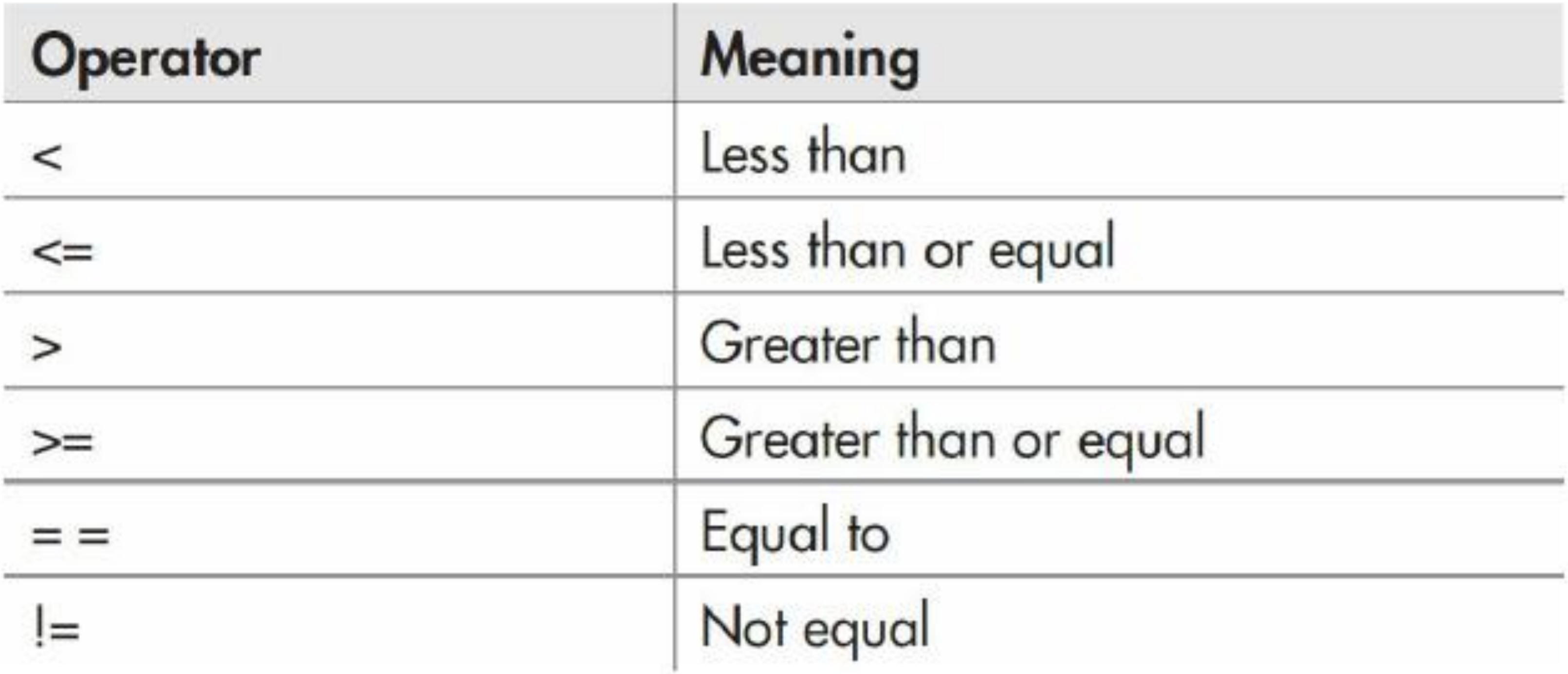
- Inside a method, execution proceeds from one statement to the next, top to bottom.
- However, it is possible to alter this flow through the use of the various program control statements supported by Java
  - The if Statement
  - Loop statements

The if Statement

- If “condition” is true next statement will be executed, otherwise it will not be executed
- If multiple statements depend on the “condition”, enclose them within a block

```
if (CONDITION)
   STATEMENT;


if (CONDITION) {
   STATEMENTS
}
```

Relational Operators



Operator	Meaning
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
= =	Equal to
!=	Not equal

if statement

```
public static void main(String[] args){
    int x=6; //also try it for 5 and 4
    if (x > 5){
        System.out.println(x + " is > 5");
    }
}
```

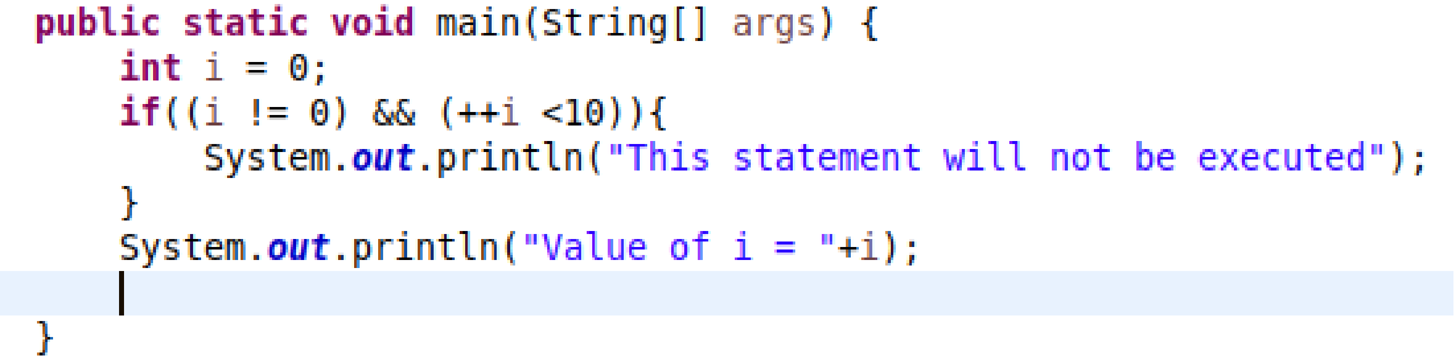
Boolean operators

&&: logical AND
||: logical OR

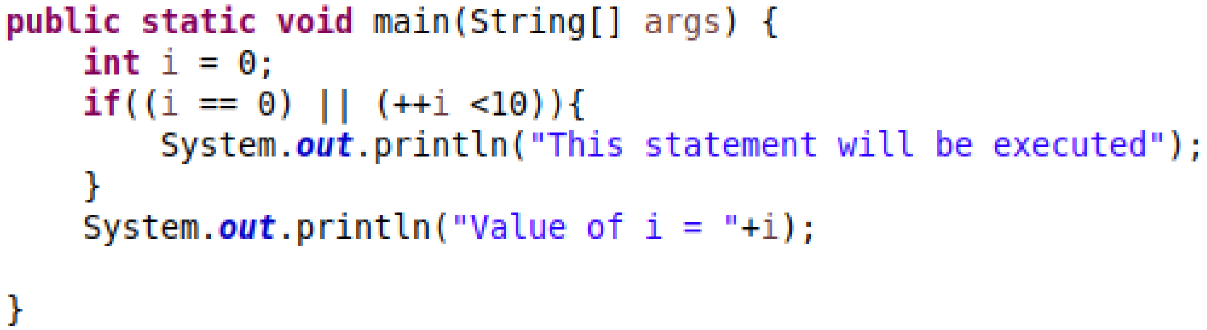
```
if (x > 6) {
    if (x < 9) {
            …
    }
}
```

```
if ( x > 6 && x < 9) {
...
}
```

What will be the value of i?



```java
public static void main(String[] args) {
    int i = 0;
    if((i != 0) && (++i <10)){
        System.out.println("This statement will not be executed");
    }
    System.out.println("Value of i = "+i);
    
}
```



```java
public static void main(String[] args) {
    int i = 0;
    if((i == 0) || (++i <10)){
        System.out.println("This statement will be executed");
    }
    System.out.println("Value of i = "+i);

}
```

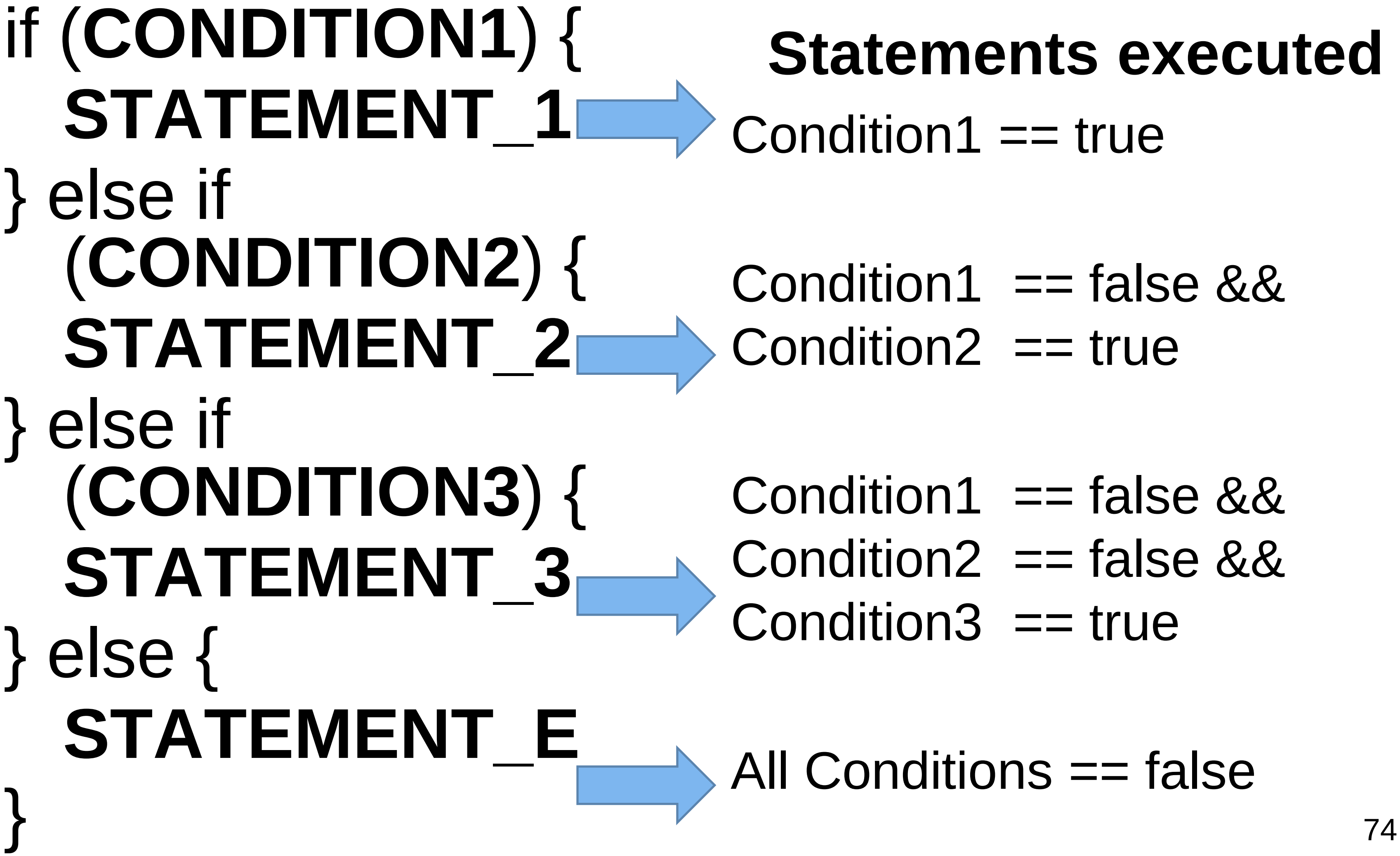
else

```
if (CONDITION) {
   STATEMENTS
} else {
   STATEMENTS
}
```

else

```
public static void main(String[] args){
    int x=6; //also try it for 5 and 4
    if (x > 5){
        System.out.println(x + " is > 5");
    } else {
        System.out.println(x + " is not > 5");
    }

}
```

else if



```
if (CONDITION1) {
   STATEMENT_1
} else if
   (CONDITION2) {
   STATEMENT_2
} else if
   (CONDITION3) {
   STATEMENT_3
} else {
   STATEMENT_E
}
```

Statements executed

Condition1 == true

Condition1 == false &&
Condition2 == true

Condition1 == false &&
Condition2 == false &&
Condition3 == true

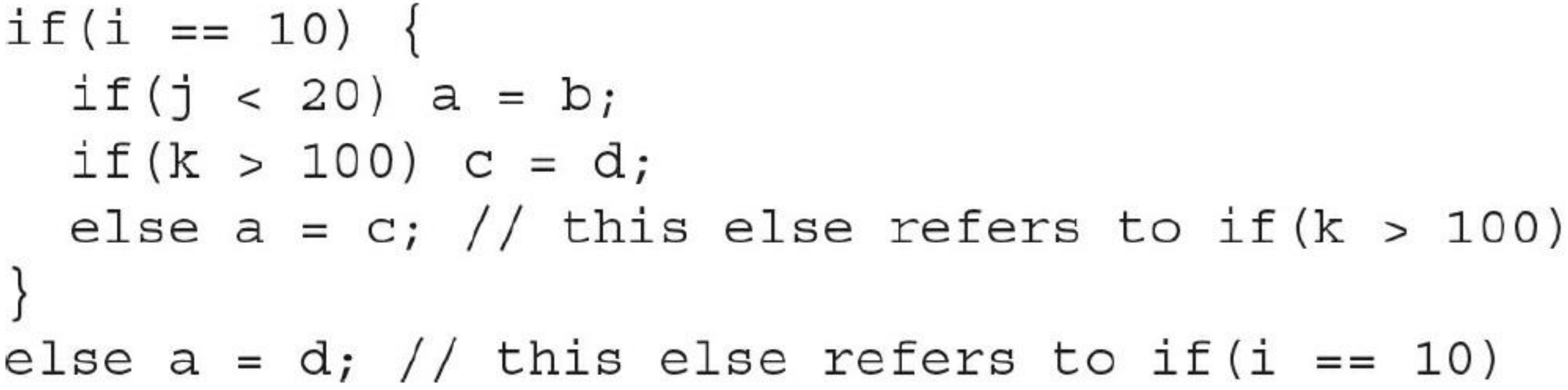
All Conditions == false

else if

```
public static void test(int x){
    }
public static void main(String[] args){
    int x=6; //also try it for 5 and 4
    if (x > 5){
            System.out.println(x + " is > 5");
    } else if (x==5) {
            System.out.println(x + " equals  5");
    } else {
            System.out.println(x + " is < 5");
    }
}
```

Nested if statements

- else statement always refers to the nearest if statement that is within the same block



```
if(i == 10) {
  if(j < 20) a = b;
  if(k > 100) c = d;
  else a = c; // this else refers to if(k > 100)
}
else a = d; // this else refers to if(i == 10)
```

The Unary Operators

+	**Unary plus operator; indicates positive value (numbers are positive without this, however)**
–	Unary minus operator; negates an expression
++	Increment operator; increments a value by 1
−−	Decrement operator; decrements a value by 1
!	Logical complement operator; inverts the value of a boolean

The Unary Operators

- The increment/decrement operators can be applied before (prefix) or after (postfix) the operand. The code

```
result++;
++result;
```

- If you are just performing a simple increment/decrement, it doesn't really matter which version you choose.

The Unary Operators

- **value = ++result;** //equivalent to

  result = result + 1;

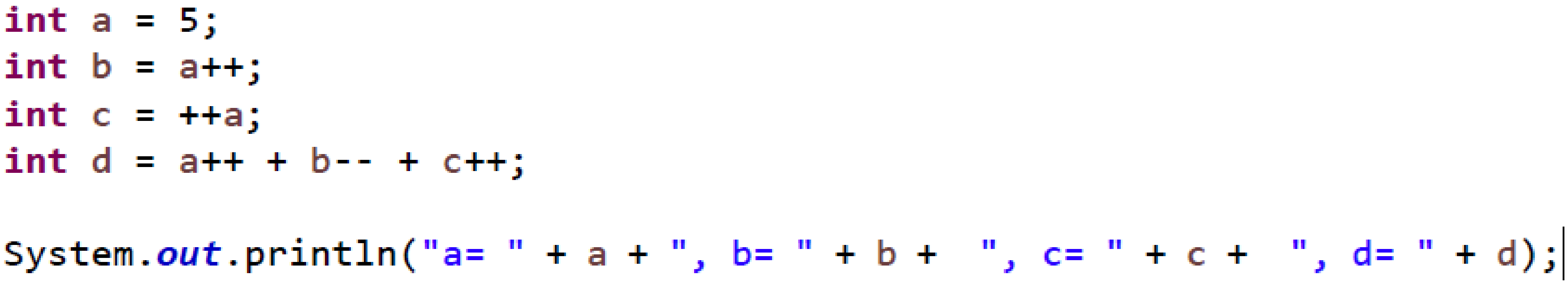
  value = result; // value gets new result
- **value = result++;** //equivalent to

  int temp = result

  result = result + 1;

  value = temp; //value gets previous result

What's the output?



```
int a = 5;
int b = a++;
int c = ++a;
int d = a++ + b-- + c++;

System.out.println("a= " + a + ", b= " + b +  ", c= " + c +  ", d= " + d);
```

References

- Dr. Öğr. Üyesi Özgür KILIÇ MSKÜ Bilgisayar Mühendisliği Bölümü
- http://math.hws.edu/javanotes/
- http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-092-introduction-to-programming-in-java-january-iap-2010/lecture-notes/
- https://www.tiobe.com/tiobe-index/
- Head First Java 2nd Edition
- Java a Beginner's guide 7th Edition