Data Structure

Lab Session 2:
Introduction to Java

U Kang

Seoul National University

Course Information

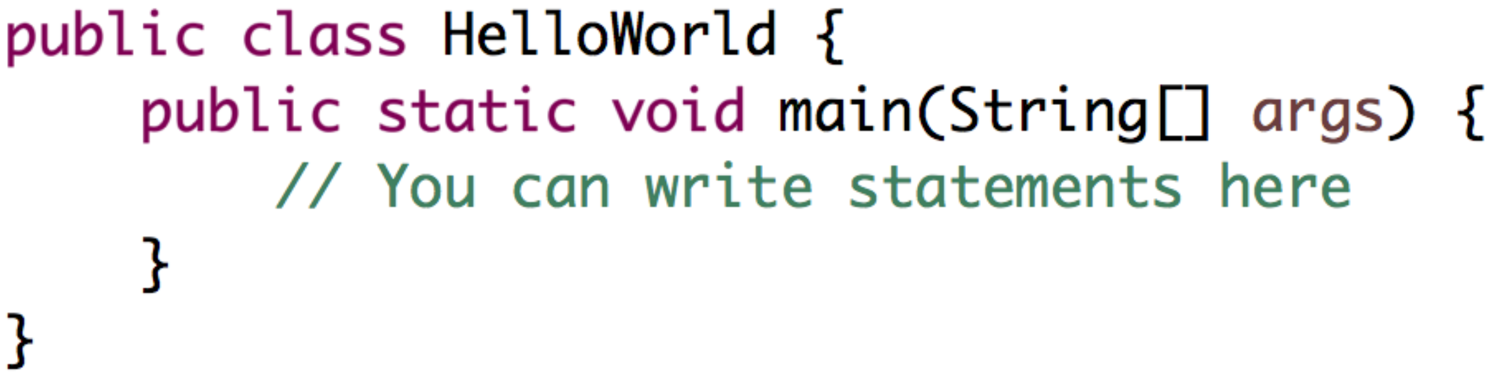
- T.A.
  - **Office 301-B119**
  - Huiwen Xu
    - E-mail: xuhuiwen33@snu.ac.kr
    - Office hour: Wed 13:00 – 14:00
  - Junghoon Kim
    - E-mail: joseph.junghoon.kim@gmail.com
    - Office hour: Thu 16:00 – 17:00
  - Seungcheol Park
    - E-mail: ant6si@snu.ac.kr
    - Office hour: Thu 14:00 – 15:00
  - Chaeheum Park (Lead T.A. for this session)
    - E-mail: chaeheum@snu.ac.kr
    - Office hour: Tue 14:00 – 15:00

Java

- One of the most popular languages
- More complex than some languages
  - Python
- Simpler than others
  - C and C++

Classes and Methods

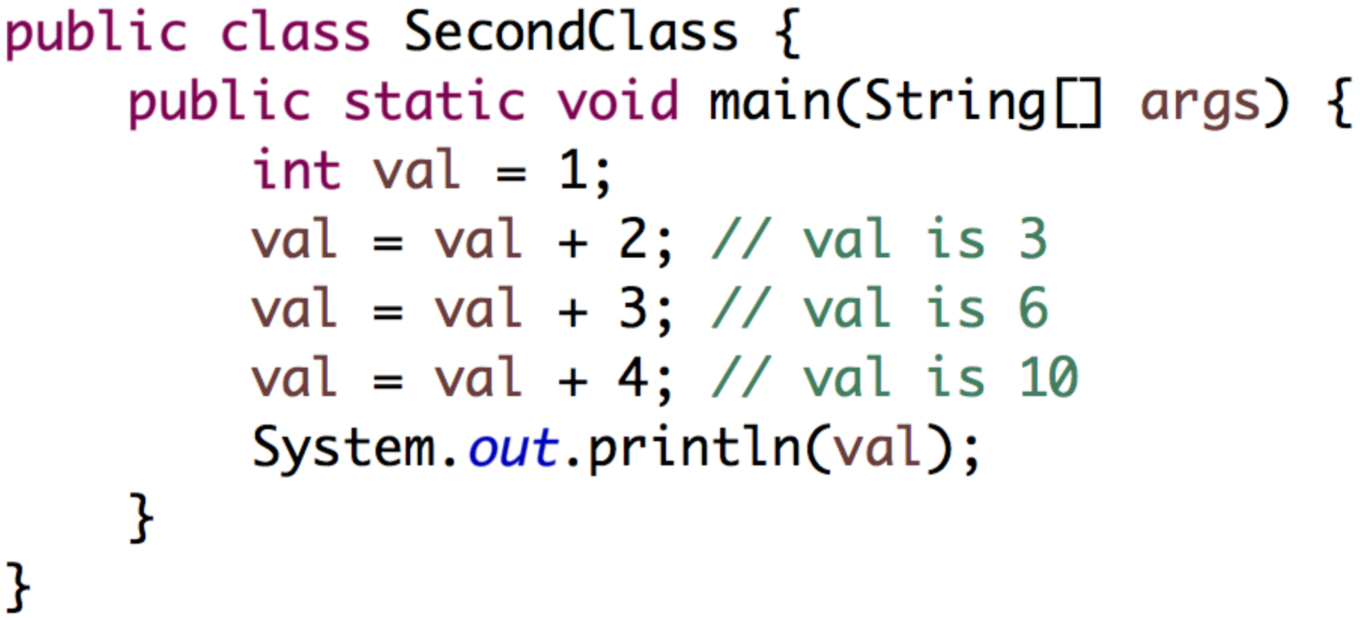
- A java file (.java) contains a class.
- Java filename and class name should be the same.
- **main** function should be defined to run.
- The figure below shows a simple program (HelloWorld.java):



```
public class HelloWorld {
    public static void main(String[] args) {
        // You can write statements here
    }
}
```

One Simple Program

- This is an example of a simple program:



```
public class SecondClass {
    public static void main(String[] args) {
        int val = 1;
        val = val + 2; // val is 3
        val = val + 3; // val is 6
        val = val + 4; // val is 10
        System.out.println(val);
    }
}
```

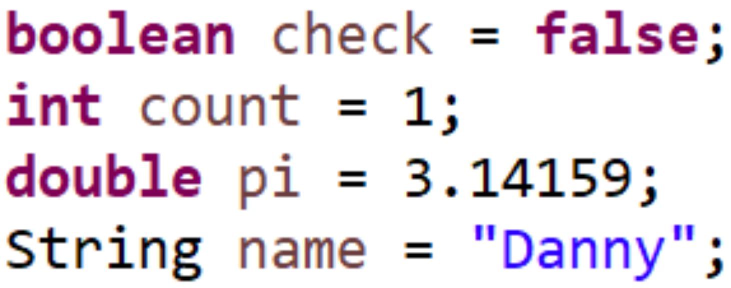
- We’ll learn how this program works.

Outline

- **Variables**
- Operators
- Conditionals
- Loops
- Methods
- Standard I/O
- Arrays
- Classes

Variables

- Named location that stores a value.
- **val** in the previous slide is a variable.
- These are examples of variables:



```
boolean check = false;
int count = 1;
double pi = 3.14159;
String name = "Danny";
```

- They store a value of a specific type.

Coding Conventions

- Coding conventions are a set of guidelines for a specific programming language
  - Whitespace, bracket placement, naming rules …
- **It is not mandatory but highly recommended**
  - We will not deduct points for not following the convention but it is a good habit
- Currently in almost any programming languages Google style guide is mainly used
  - Google style guide for Java [link]

Reformat Code

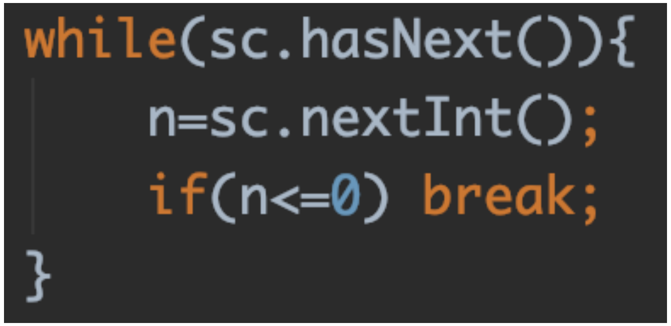
Details

- In IntelliJ IDEA most of the conventions (whitespace, bracket positioning, line wrapping) are automatically covered
  - <Code> - <Reformat Code>
  - (Window) Ctrl + Alt + L
  - (Mac) Command + Option +L

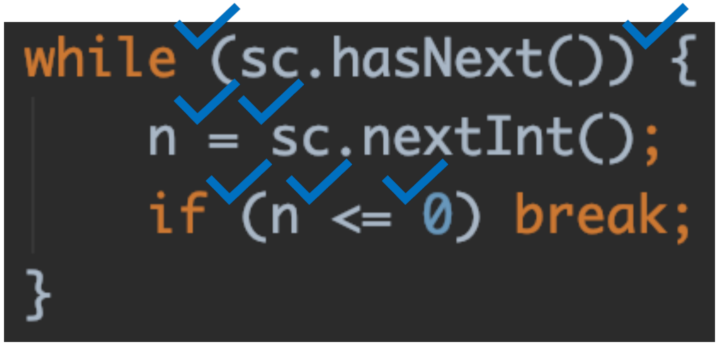
Reformat Code

Details

- Example in IntelliJ IDEA
  - The checkmark indicates where it has been changed



```
while(sc.hasNext()){
    n=sc.nextInt();
    if(n<=0) break;
}
```

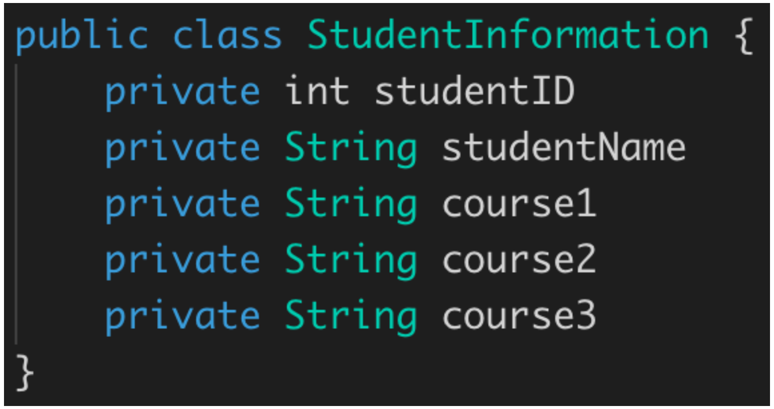


```
while (sc.hasNext()) {
    n = sc.nextInt();
    if (n <= 0) break;
}
```

  - <Reformat Code> feature automatically improves code readability and consistency

Variable Naming Rules

- A variable’s name is a string of English characters and digits
- Variable names are case-sensitive
- The name should use `lowerCamelCase` format
  - `lowerCamelCase` : First word is lowercase and all following words start with an UPPERCASE
  - In abbreviations all characters are UPPERCASE

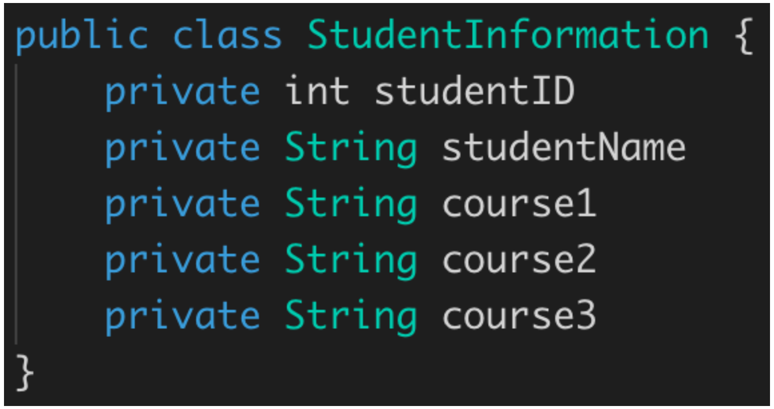


```
public class StudentInformation {
    private int studentID
    private String studentName
    private String course1
    private String course2
    private String course3
}
```

Camel Case

Details

- lowerCamelCase
  - First word is lowercase and all following words start with an UPPERCASE
  - Ex) studentName, studentID
- UpperCamelCase
  - All words start with an UPPERCASE
  - Ex) StudentInformation
- In abbreviations all characters are UPPERCASE



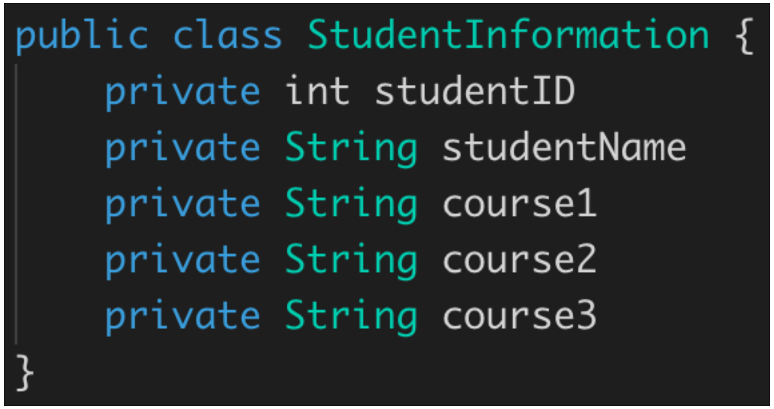
```
public class StudentInformation {
    private int studentID
    private String studentName
    private String course1
    private String course2
    private String course3
}
```

Convention Java vs Python

Details

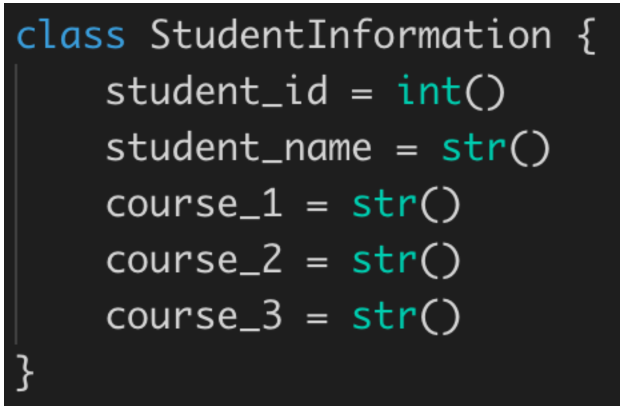
- Java uses CamelCase for naming convention
  - `StudentInformation, studentName`
- Python uses underlines for naming convention
  - `student_information, student_name`

Java



```
public class StudentInformation {
    private int studentID
    private String studentName
    private String course1
    private String course2
    private String course3
}
```

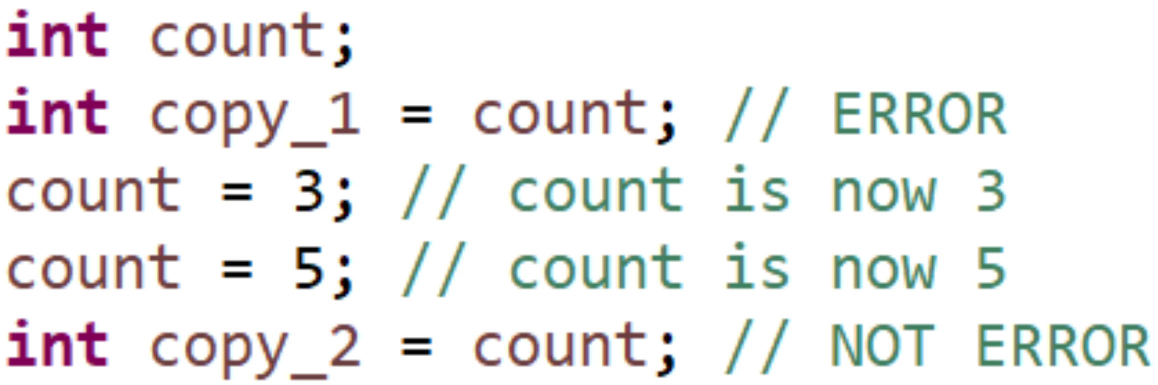
Python



```
class StudentInformation {
    student_id = int()
    student_name = str()
    course_1 = str()
    course_2 = str()
    course_3 = str()
}
```

Assignments

- Values are **assigned** after variables are **declared**.
- Assignment operator (=) is used.
- Values cannot be used without assignments.



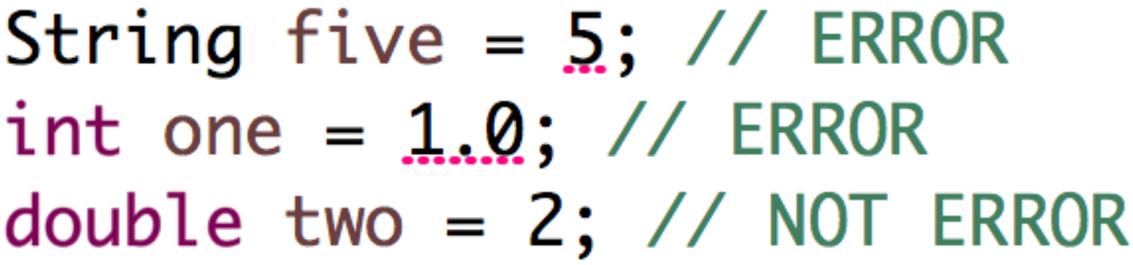
```
int count;
int copy_1 = count; // ERROR
count = 3; // count is now 3
count = 5; // count is now 5
int copy_2 = count; // NOT ERROR
```

Types

- Kinds of values that can be stored.
- There are primitive types for Java:
  - **boolean**: a truth value (**true** or **false**).
  - **int**: an integer (0, 1, 10, …)
  - **float** or **double**: a real number (3.1415, -1.0, …)
  - **char**: a character (‘a’, ‘b’, …)
  - **String**: a text (“hello”, “example”, …)

Mismatched Types

- Java verifies that types always match.

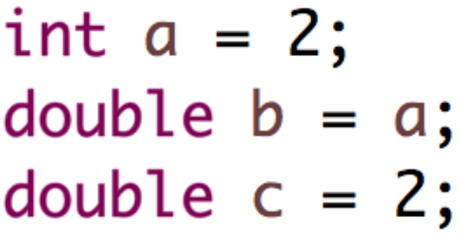


```
String five = 5; // ERROR
int one = 1.0; // ERROR
double two = 2; // NOT ERROR
```

- These errors are reported before executions.
- Note that the 3rd line is not an error:
  - it is called as a type conversion.

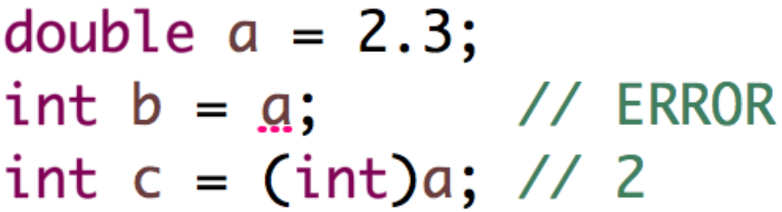
Type Conversions (1)

- int can be converted into double **implicitly**.



```
int a = 2;
double b = a;
double c = 2;
```

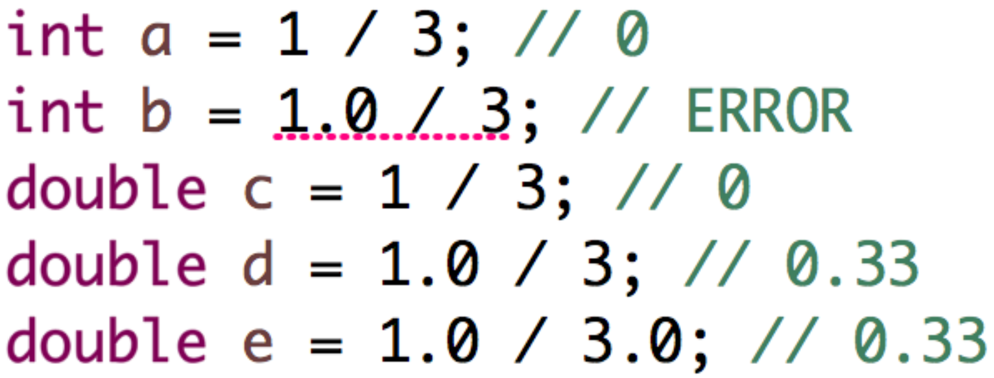
- double can be converted into int only **explicitly**.



```
double a = 2.3;
int b = a;         // ERROR
int c = (int)a; // 2
```

Type Conversions (2)

- Constant values in a code have types too.
- They are declared and converted implicitly.



```
int a = 1 / 3; // 0
int b = 1.0 / 3; // ERROR
double c = 1 / 3; // 0
double d = 1.0 / 3; // 0.33
double e = 1.0 / 3.0; // 0.33
```

Outline

- Variables
- **Operators**
- Conditionals
- Loops
- Methods
- Standard I/O
- Arrays
- Classes

Math Operators

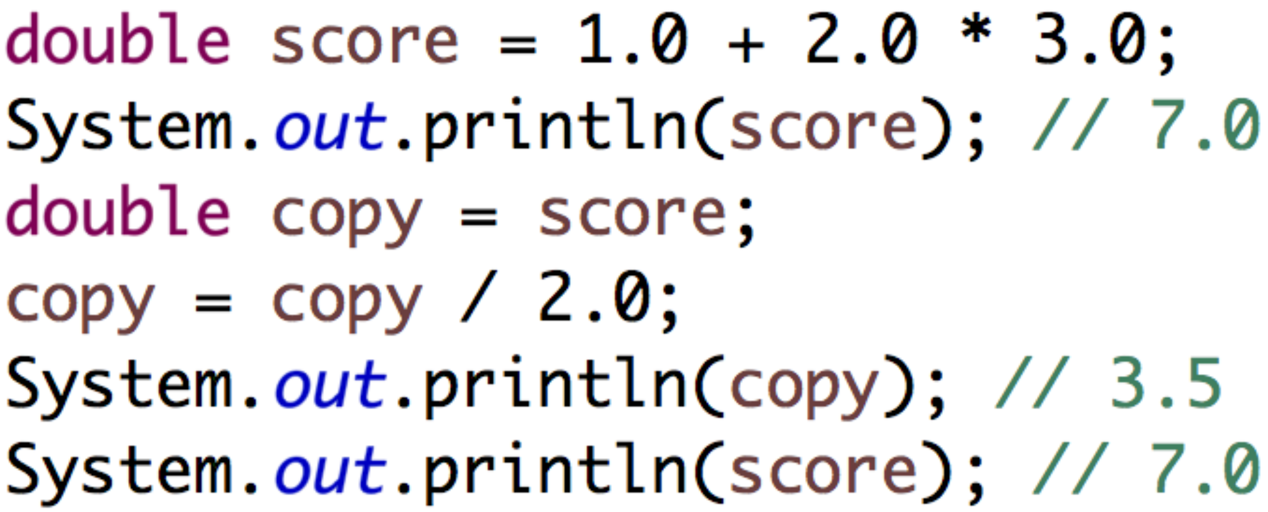
- Symbols that perform single computations.
- These are math operators in Java:
  - Addition (+)
  - Subtraction (-)
  - Multiplication (*)
  - Division (/)
- Assignment (=) is also an operator.
  - although it is not included in math operators.

Order of Operations

- Follows standard order of operations:
  1. Parentheses
  2. Multiplication and division
  3. Addition and subtraction
- Precedence like math, left to right.

Example of Operators (1)

- Math operators are used like this:

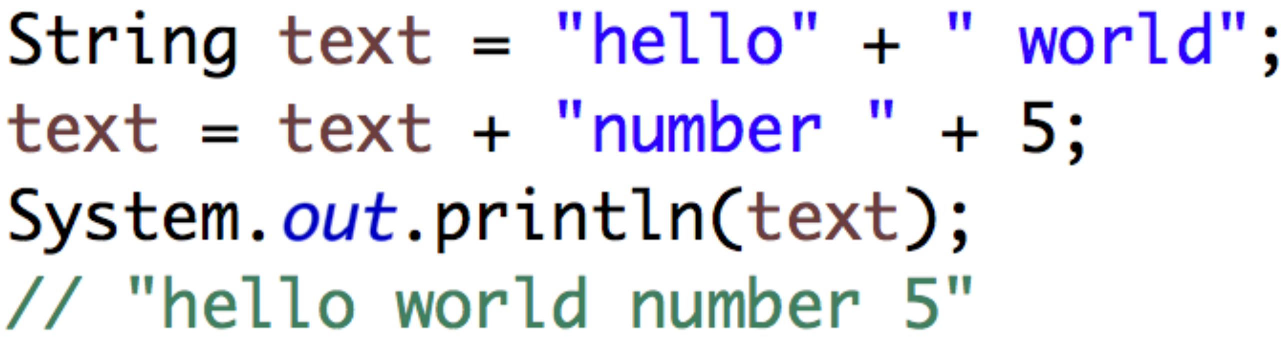


```
double score = 1.0 + 2.0 * 3.0;
System.out.println(score); // 7.0
double copy = score;
copy = copy / 2.0;
System.out.println(copy); // 3.5
System.out.println(score); // 7.0
```

- Look at the differences from the previous one.

String Concatenation

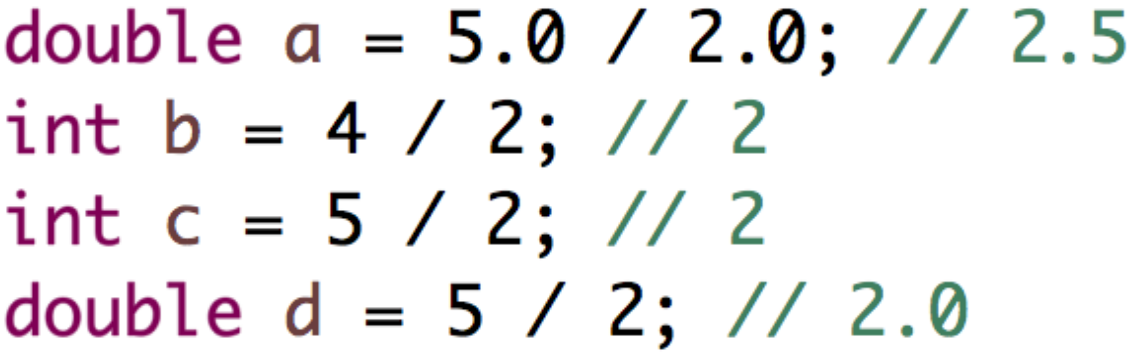
- (+) is defined for several data types:
  - **int**: addition for integers.
  - **double**: addition for real values.
  - **String**: concatenation for texts.



```
String text = "hello" + " world";
text = text + "number " + 5;
System.out.println(text);
// "hello world number 5"
```

Division Operator

- Division (/) operates differently on 2 types:
  - integers and doubles.
  - because integers cannot store real numbers.
- See the example below:



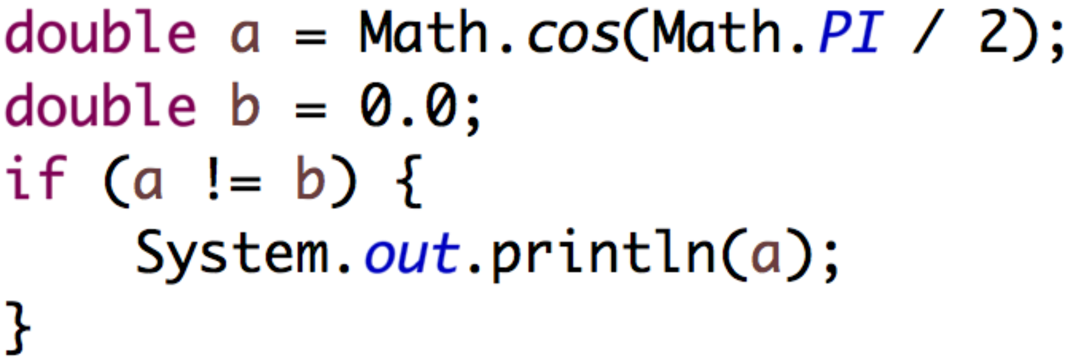
```
double a = 5.0 / 2.0; // 2.5
int b = 4 / 2; // 2
int c = 5 / 2; // 2
double d = 5 / 2; // 2.0
```

Comparison Operators

- There are comparison operators in Java:
  - x > y means x is greater than y.
  - x < y means x is smaller than y.
  - x >= y means x is greater than or equal to y.
  - x <= y means x is smaller than or equal to y.
  - x == y means x is equal to y.
  - x != y means x is not equal to y.
- Their results are values of boolean types.
- Equality (==) is different from assignment (=).

Comparison for Real Numbers

- Do NOT call equality (==) on doubles.
- See the example below:
  - **Math.cos()** is a function to compute a cosine.
  - **Math.PI** is a constant having 3.1415…
  - The result is 6.123233995736766E-17.



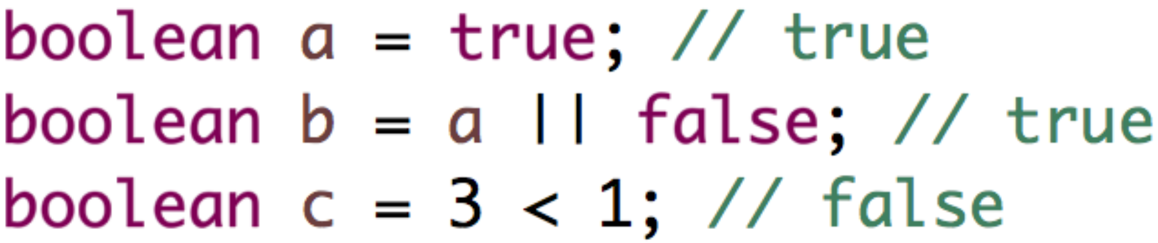
```
double a = Math.cos(Math.PI / 2);
double b = 0.0;
if (a != b) {
    System.out.println(a);
}
```

Boolean Operators

- These are boolean operators:
  - && means logical AND.
  - || means logical OR.
- See these examples:
  - $x > 1$ && $y > 1$ means x and y is greater than 1.
  - $x > y$ || $x > z$ means x is greater than y or z.

Example of Operators (2)

- Boolean operators can be used like this:



```
boolean a = true; // true
boolean b = a || false; // true
boolean c = 3 < 1; // false
```

- In fact, they are very important in later sections.

Outline

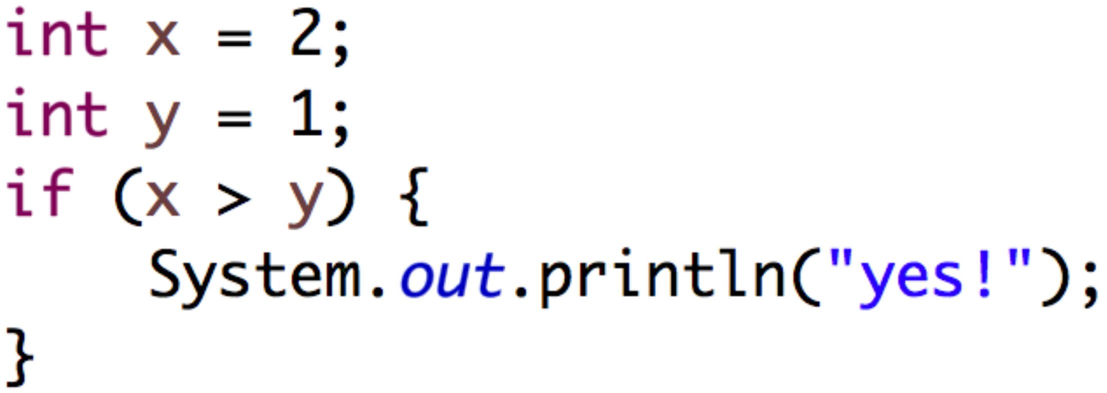
- Variables
- Operators
- **Conditionals**
- Loops
- Methods
- Standard I/O
- Arrays
- Classes

Conditionals

- Conditionals are essential to programs.
  - They control flows of a program.
  - They determine what to do based on current states.
- There are 3 conditional statements:
  - **if**, **else**, and **else if**.
- They are used with comparison operators.

if statement

- **if** statement is used like this:

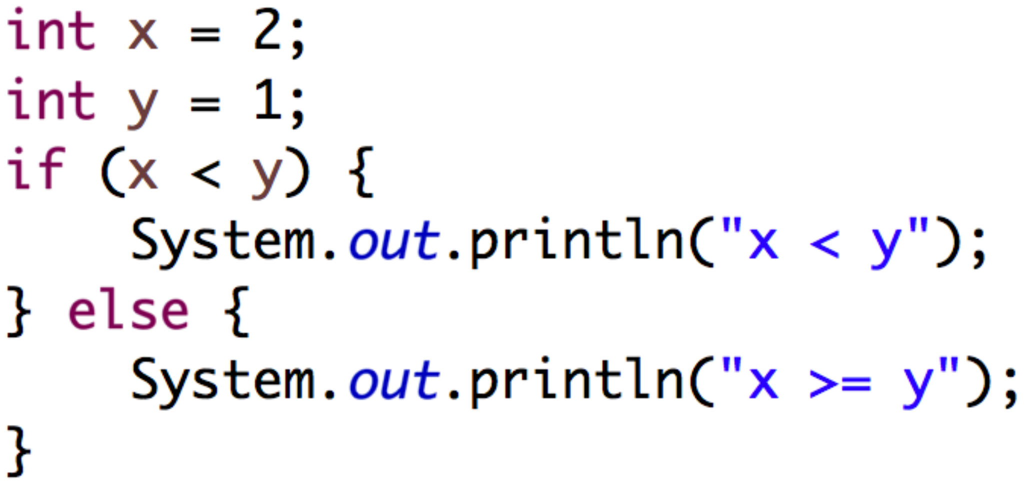


```
int x = 2;
int y = 1;
if (x > y) {
    System.out.println("yes!");
}
```

- The print statement is executed
  - because x is greater than y.

else statement

- **else** statement is used like this:

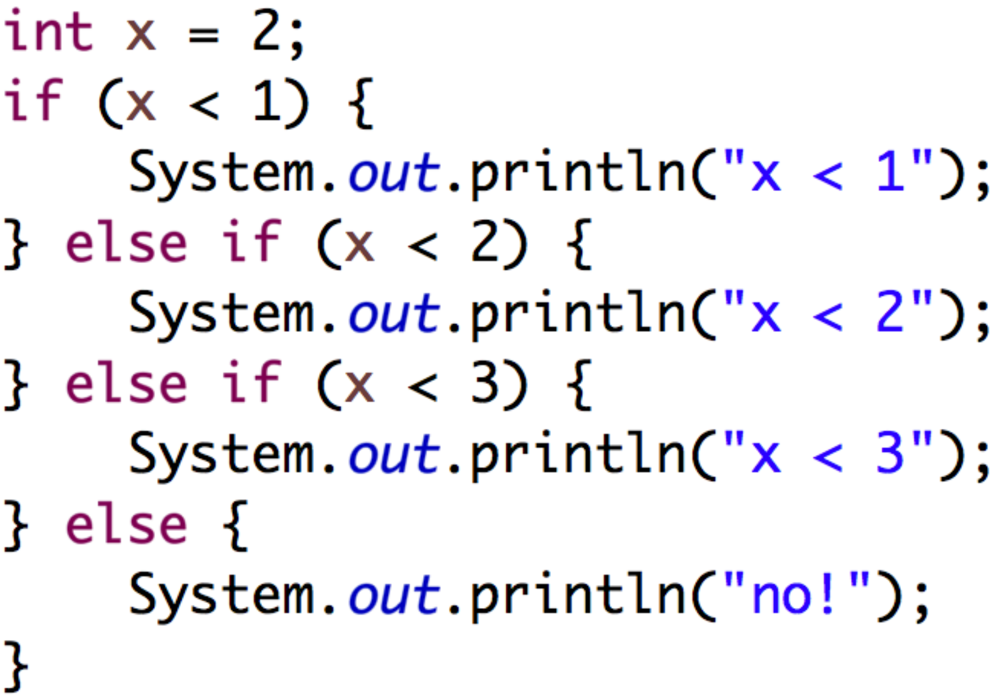


```
int x = 2;
int y = 1;
if (x < y) {
    System.out.println("x < y");
} else {
    System.out.println("x >= y");
}
```

- The print statement in **else** is executed
  - because x is not smaller than y.

else if statement

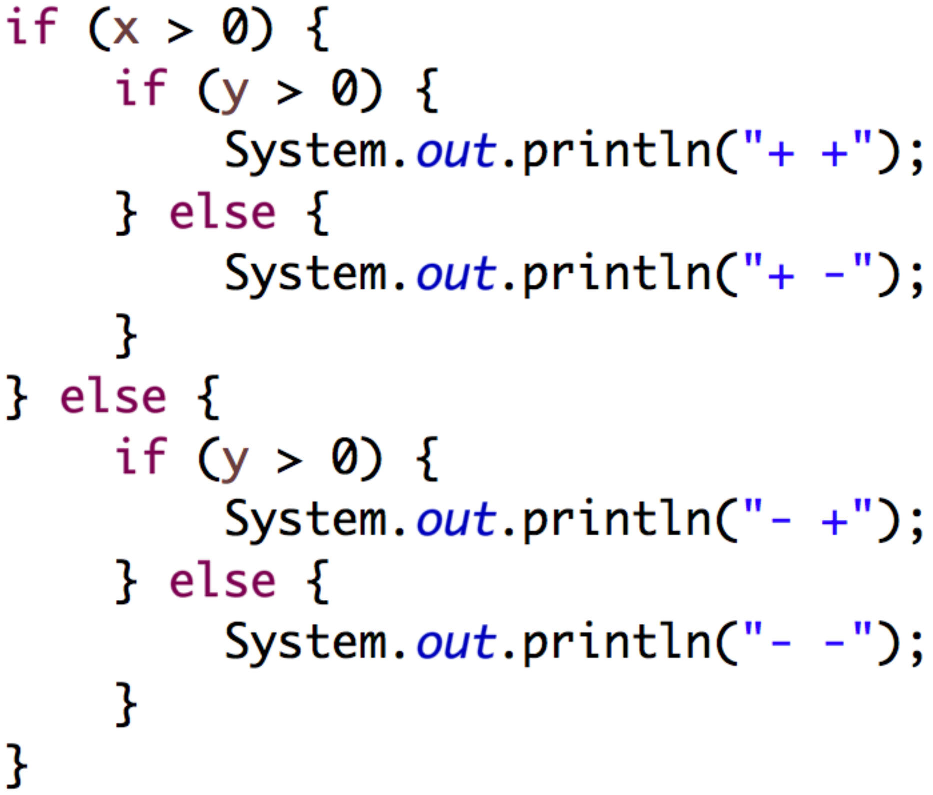
- **else if** statements are used like this:



```
int x = 2;
if (x < 1) {
    System.out.println("x < 1");
} else if (x < 2) {
    System.out.println("x < 2");
} else if (x < 3) {
    System.out.println("x < 3");
} else {
    System.out.println("no!");
}
```

Examples of Conditionals (1)

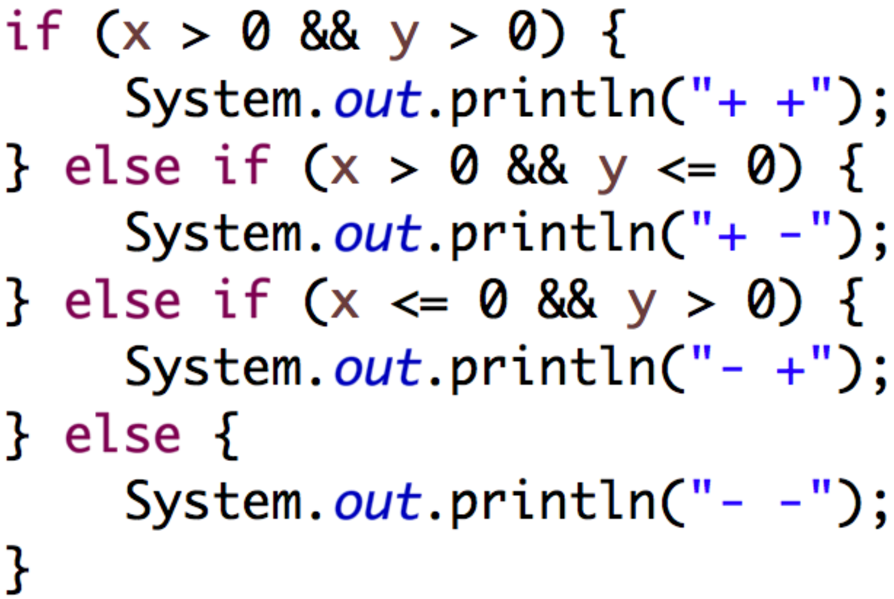
- Conditionals can include other conditionals.



```
if (x > 0) {
    if (y > 0) {
        System.out.println("+ +");
    } else {
        System.out.println("+ -");
    }
} else {
    if (y > 0) {
        System.out.println("- +");
    } else {
        System.out.println("- -");
    }
}
```

Example of Conditionals (2)

- This is the same as the previous example
  - which is written differently.



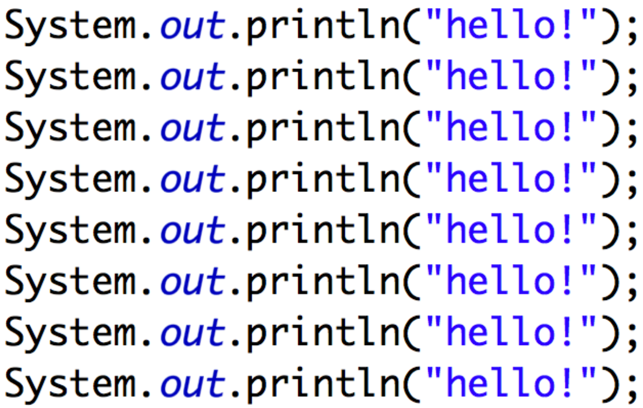
```
if (x > 0 && y > 0) {
    System.out.println("+ +");
} else if (x > 0 && y <= 0) {
    System.out.println("+ -");
} else if (x <= 0 && y > 0) {
    System.out.println("- +");
} else {
    System.out.println("- -");
}
```

Outline

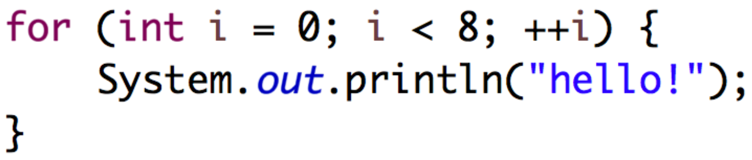
- Variables
- Operators
- Conditionals
- **Loops**
- Methods
- Standard I/O
- Arrays
- Classes

Loops

- Loops are used in programming to **repeat a specific block** until some end condition is met.
- Both code blocks below have same output:



```java
System.out.println("hello!");
System.out.println("hello!");
System.out.println("hello!");
System.out.println("hello!");
System.out.println("hello!");
System.out.println("hello!");
System.out.println("hello!");
System.out.println("hello!");
```

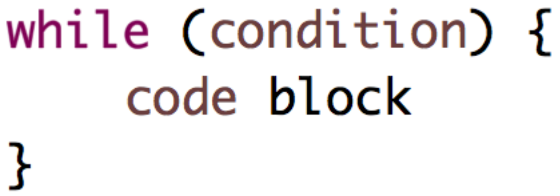


```java
for (int i = 0; i < 8; ++i) {
    System.out.println("hello!");
}
```

- There are several loop operators in Java.

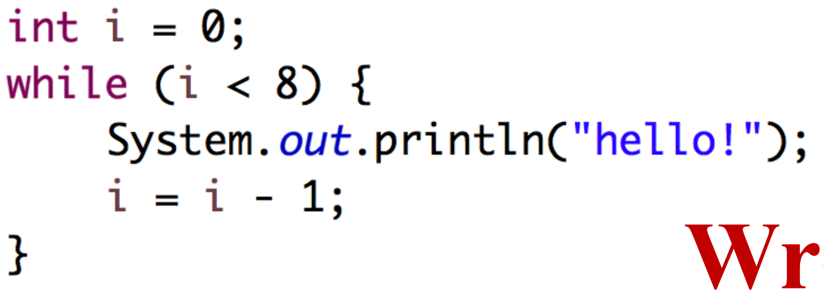
The ‘while’ Loop

- The ‘while’ statement repeats a code block until `condition` is false.



```
while (condition) {
      code block
}
```

- You should be careful about infinite loop.

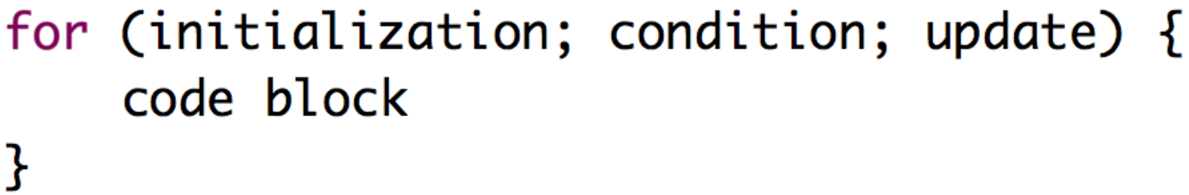


```
int i = 0;
while (i < 8) {
     System.out.println("hello!");
     i = i - 1;
}
```

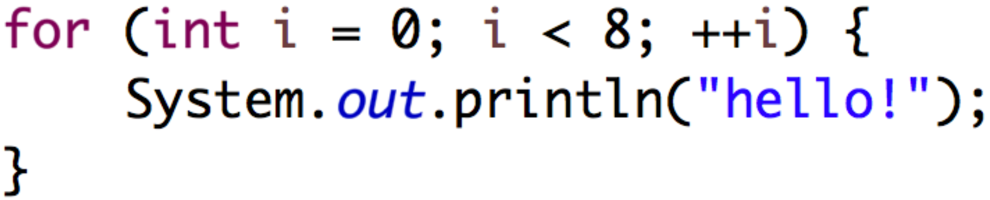
Wrong!

The ‘for’ Loop

- The ‘for’ statement is the same as ’while’ operator but it has two convenient features:
  - Initialization
  - Update



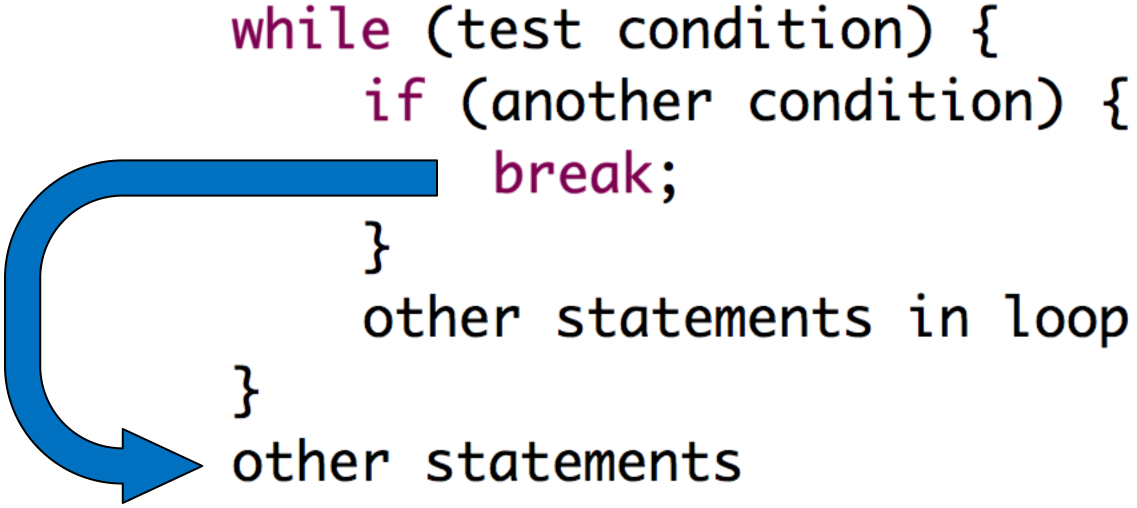
```
for (initialization; condition; update) {
      code block
}
```



```
for (int i = 0; i < 8; ++i) {
      System.out.println("hello!");
}
```

‘break’ Statement

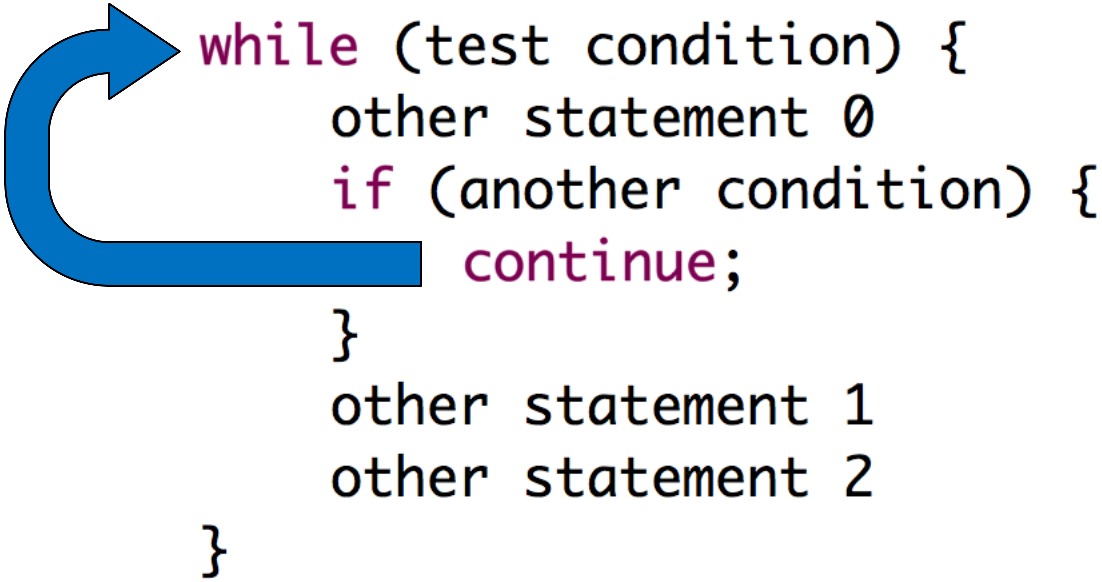
- Sometimes we need to terminate loop immediately without checking the test condition.
- The ’break’ statement terminates the loop immediately when it is encountered.



```
while (test condition) {
    if (another condition) {
        break;
    }
    other statements in loop
}
other statements
```

‘continue’ Statement

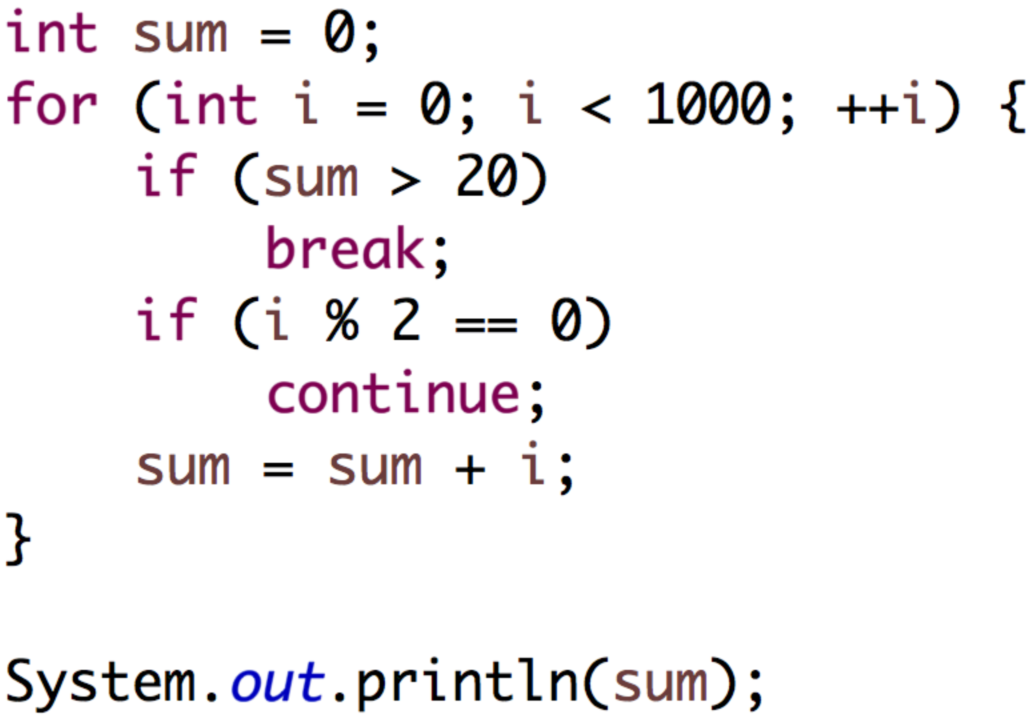
- The ‘continue’ statement skip all statements remaining in current loop.



```
while (test condition) {
    other statement 0
    if (another condition) {
        continue;
    }
    other statement 1
    other statement 2
}
```

Loop Example

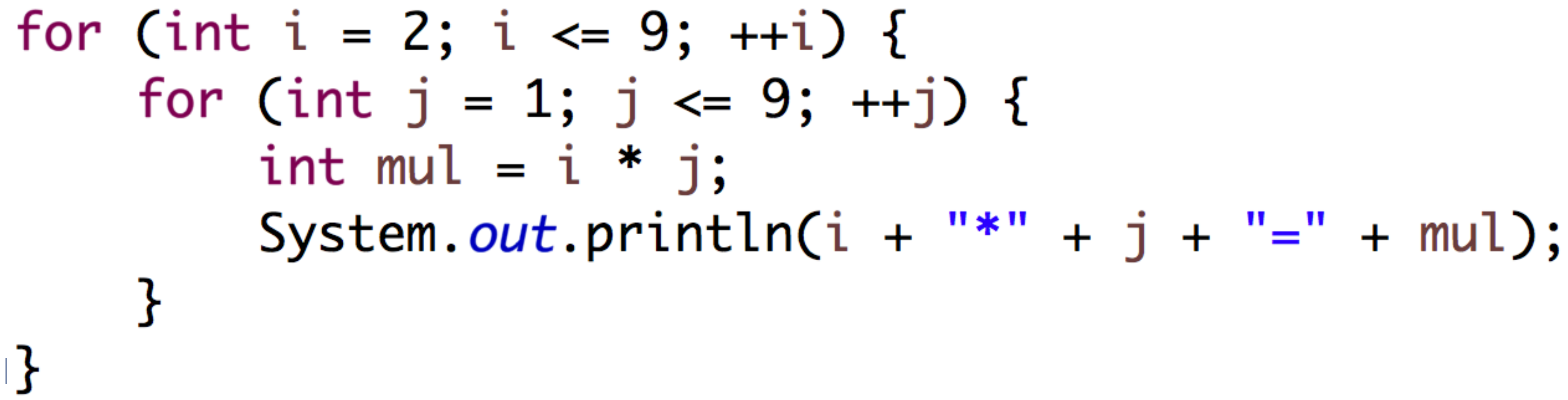
- What is the output value of below program?



```
int sum = 0;
for (int i = 0; i < 1000; ++i) {
    if (sum > 20)
        break;
    if (i % 2 == 0)
        continue;
    sum = sum + i;
}

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

Nested Loops

- A loop can be in another loop.
  - We call this ‘nested loop’ or ‘embedded loop’.



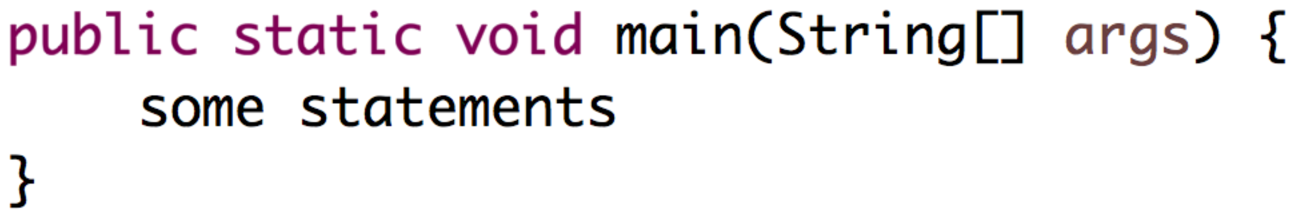
```
for (int i = 2; i <= 9; ++i) {
    for (int j = 1; j <= 9; ++j) {
        int mul = i * j;
        System.out.println(i + "*" + j + "=" + mul);
    }
}
```

Outline

- Variables
- Operators
- Conditionals
- Loops
- **Methods**
- Standard I/O
- Arrays
- Classes

Methods

- A method is a collection of statements that are grouped together to perform an operation.
- We’ve already created a method!

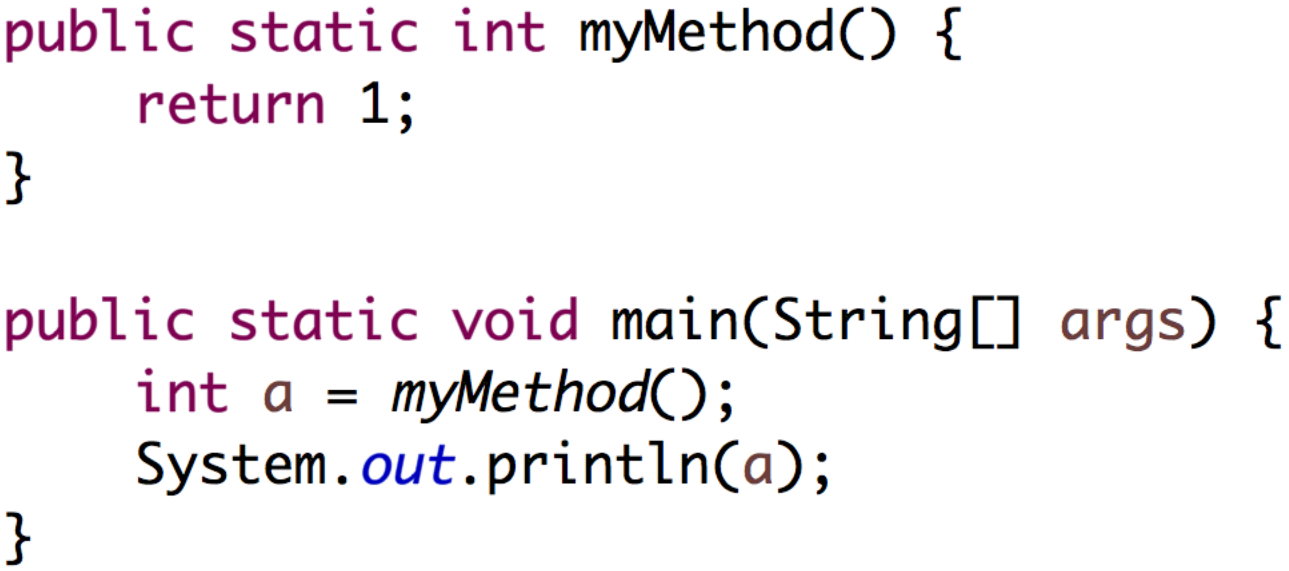


```
public static void main(String[] args) {
    some statements
}
```

- Method can have several parameters and a returned value.

Calling a Method (1)

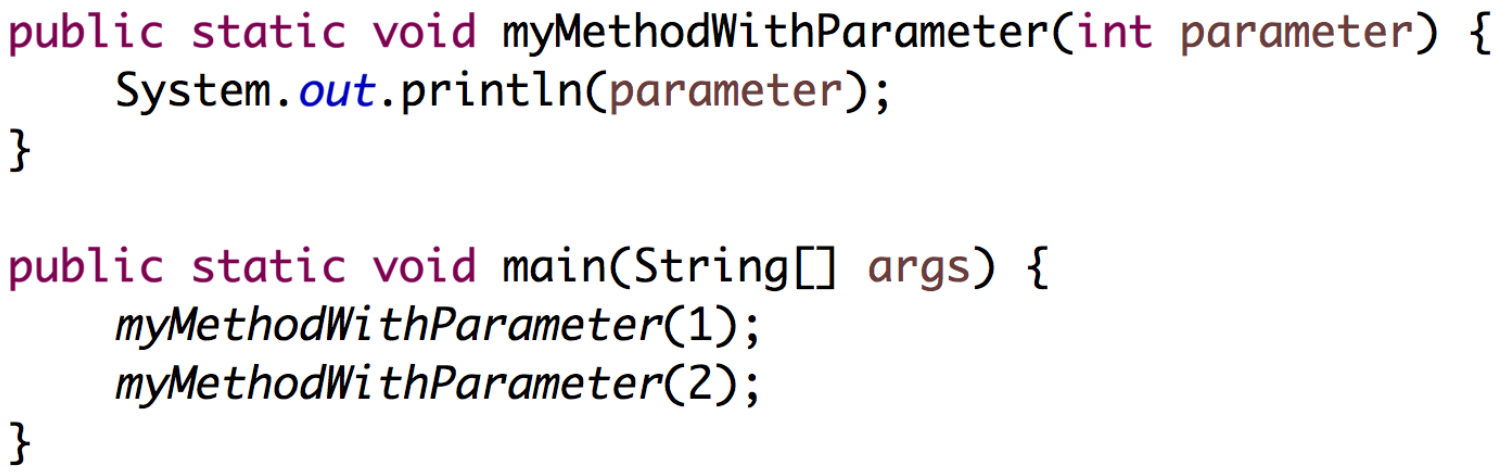
- You can call a method by the name of the method with parenthesis.



```java
public static int myMethod() {
    return 1;
}

public static void main(String[] args) {
    int a = myMethod();
    System.out.println(a);
}
```

Calling a Method (2)

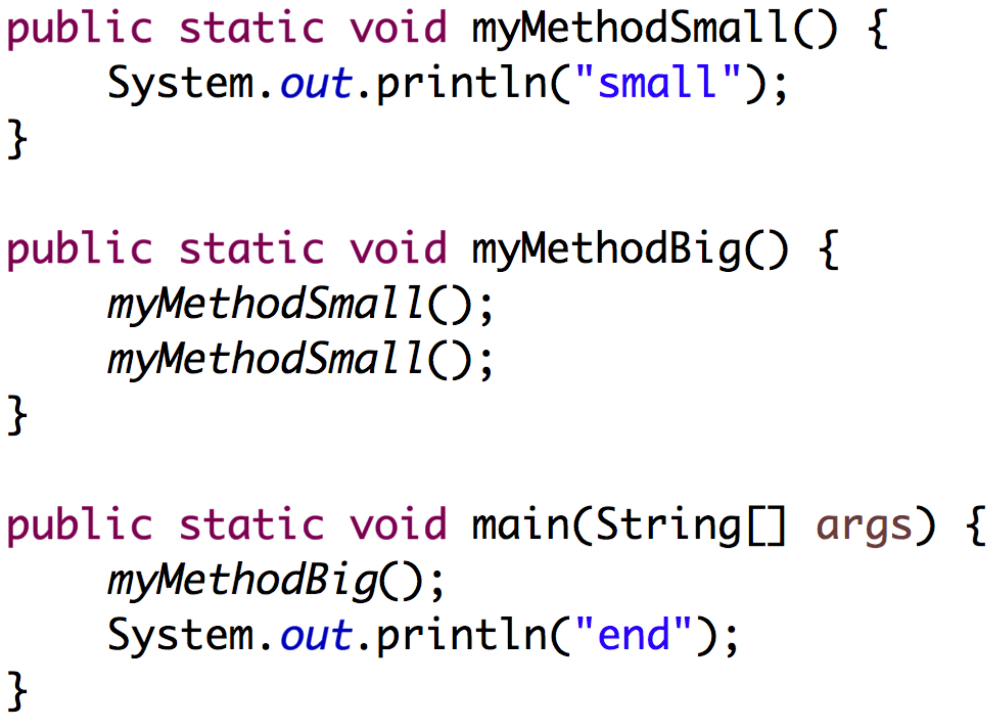
- You can pass parameters by adding values in the parenthesis.



```
public static void myMethodWithParameter(int parameter) {
    System.out.println(parameter);
}

public static void main(String[] args) {
    myMethodWithParameter(1);
    myMethodWithParameter(2);
}
```

Execution Path of Methods

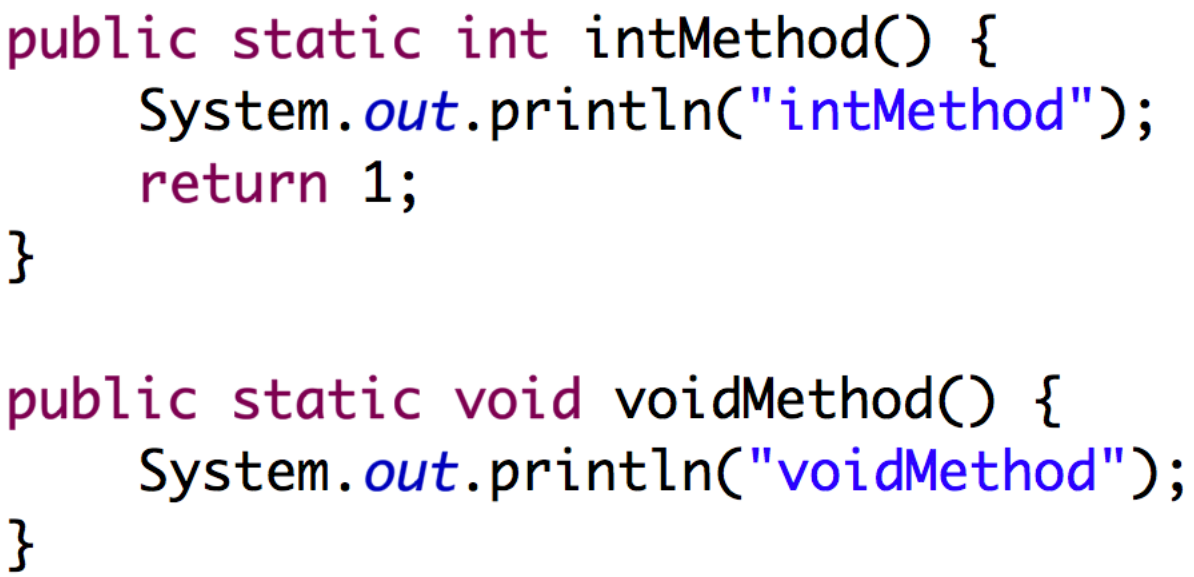
- After a callee method is done, the next statement in the caller method will be executed.



```
        public static void myMethodSmall() {
(5) (3)     System.out.println("small");
        }

        public static void myMethodBig() {
    (2)     myMethodSmall();
    (4)     myMethodSmall();
        }

        public static void main(String[] args) {
    (1)     myMethodBig();
    (5)     System.out.println("end");
        }
```

‘return’ Statement

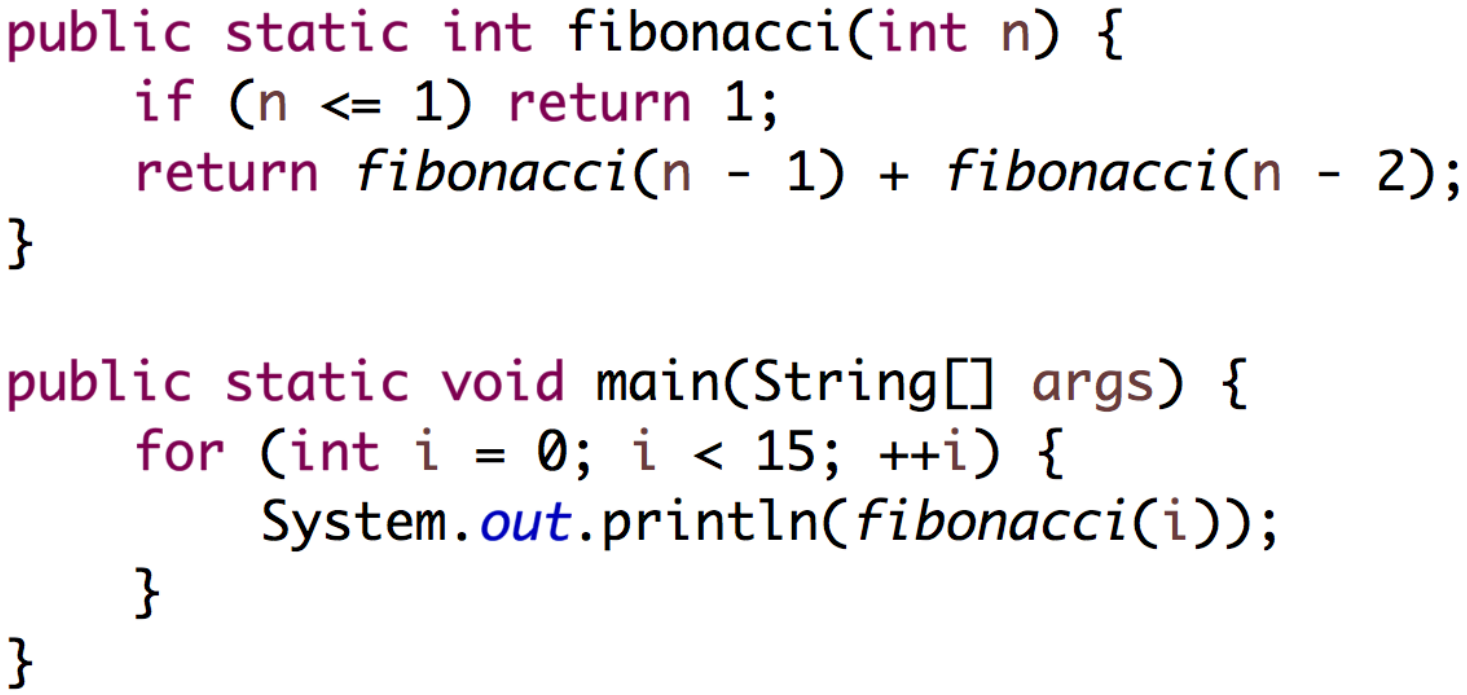
- The ‘return’ statement returns a value to a caller and exits the method
  - If the return type is ‘void’, we can omit ’return’ statement.



```
public static int intMethod() {
    System.out.println("intMethod");
    return 1;
}

public static void voidMethod() {
    System.out.println("voidMethod");
}
```

Recursive Method

- Method can call itself (recursive method).

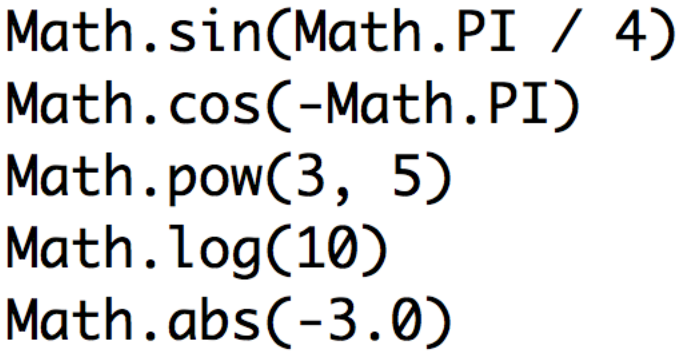


```
public static int fibonacci(int n) {
    if (n <= 1) return 1;
    return fibonacci(n - 1) + fibonacci(n - 2);
}

public static void main(String[] args) {
    for (int i = 0; i < 15; ++i) {
        System.out.println(fibonacci(i));
    }
}
```

- You should be careful about terminate condition!

Built-in Methods

- Some useful methods are provided:
  - E.g. mathematical functions



```
Math.sin(Math.PI / 4)
Math.cos(-Math.PI)
Math.pow(3, 5)
Math.log(10)
Math.abs(-3.0)
```

Methods as Building Blocks

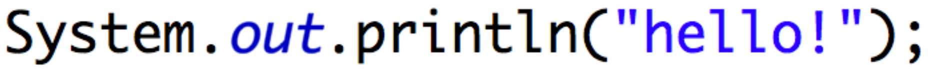
- Methods are building blocks of program.
  - Many programs are built on multiple methods.
- Methods can be individually developed and reused.
- Methods are like black boxes.
  - Users do not need to know details of the methods.

Outline

- Variables
- Operators
- Conditionals
- Loops
- Methods
- **Standard I/O**
- Arrays
- Classes

Standard I/O

- Java provides some classes and methods to input and output values.
- We’ve already seen one of output features.



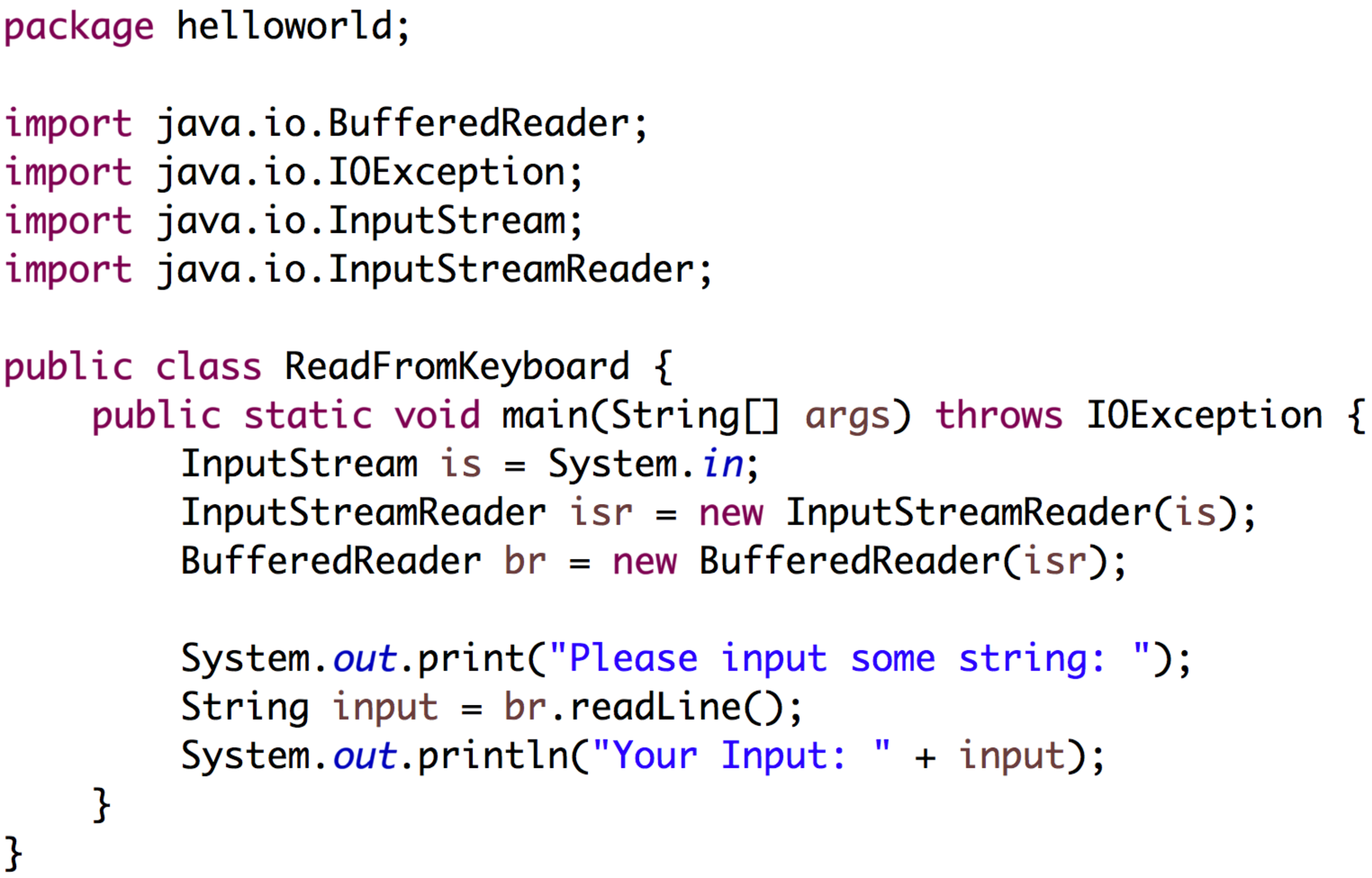
```
System.out.println("hello!");
```

- In Java, you can read and write values with various resources.
  - InputStream – an abstraction of input resources
  - OutputStream – an abstraction of output resources

InputStream

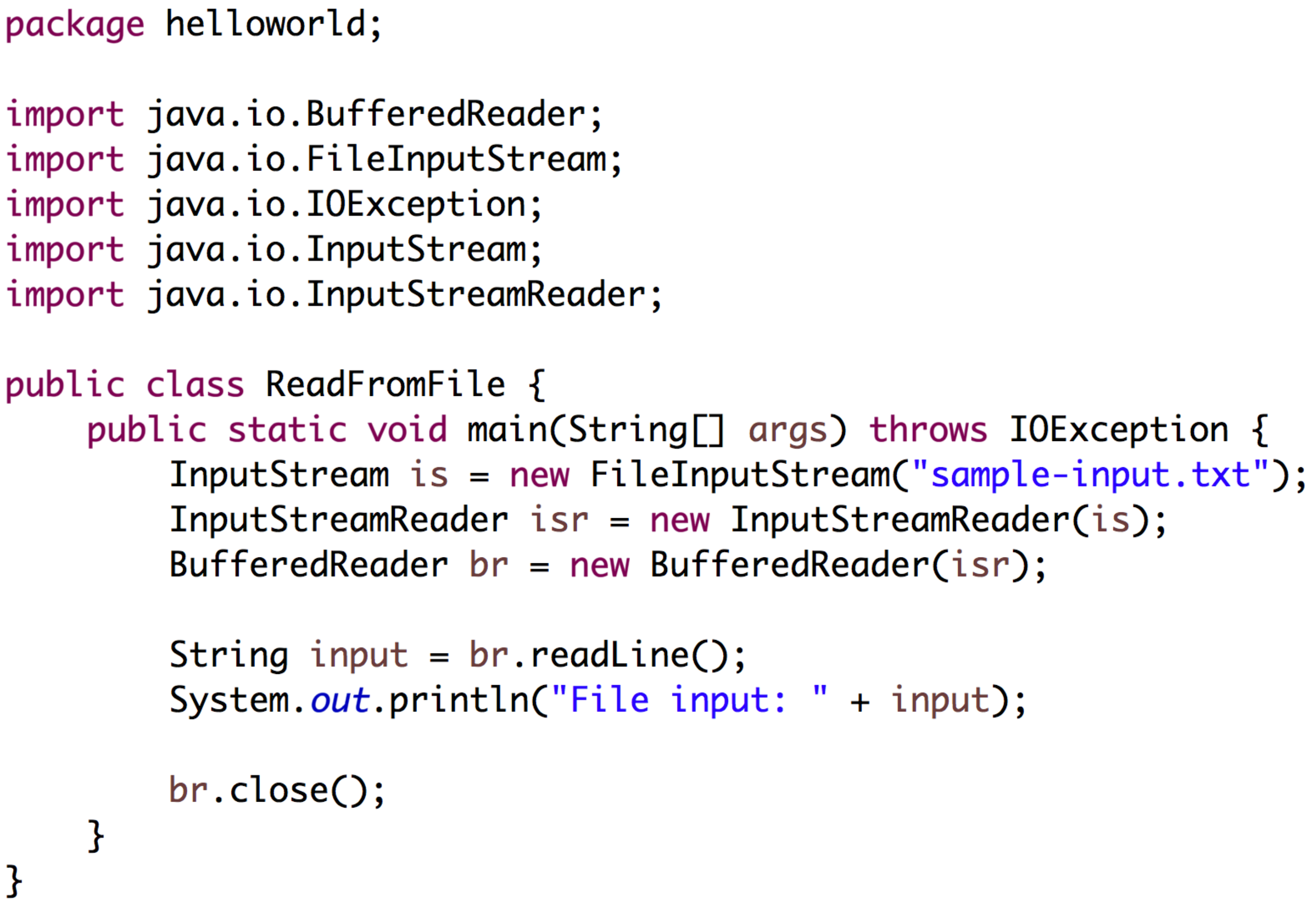
- ‘InputStream’ is an abstraction of a stream of bytes to be read.
  - System.in → an input stream from keyboard
  - FileInputStream → an input stream from file
  - SocketInputStream → an input stream from network
- A byte is just a number.
  - Bytes can be interpreted as characters, numbers, etc..
  - ‘Reader’ classes interpret the stream of bytes.

Read a String from Keyboard



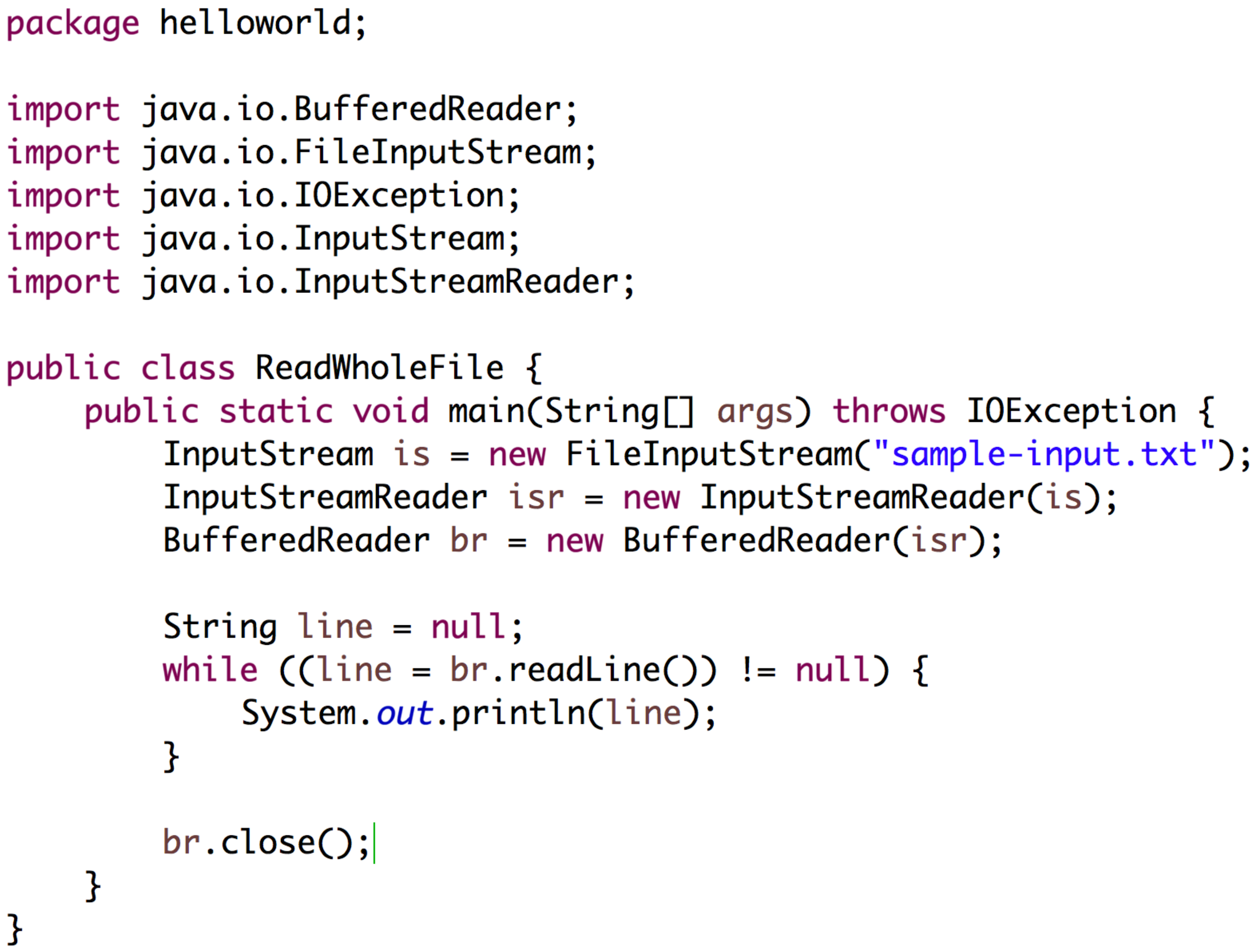
```
package helloworld;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

public class ReadFromKeyboard {
    public static void main(String[] args) throws IOException {
        InputStream is = System.in;
        InputStreamReader isr = new InputStreamReader(is);
        BufferedReader br = new BufferedReader(isr);

        System.out.print("Please input some string: ");
        String input = br.readLine();
        System.out.println("Your Input: " + input);
    }
}
```

Read a String from File



```
package helloworld;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

public class ReadFromFile {
    public static void main(String[] args) throws IOException {
        InputStream is = new FileInputStream("sample-input.txt");
        InputStreamReader isr = new InputStreamReader(is);
        BufferedReader br = new BufferedReader(isr);

        String input = br.readLine();
        System.out.println("File input: " + input);

        br.close();
    }
}
```

Read a Whole File

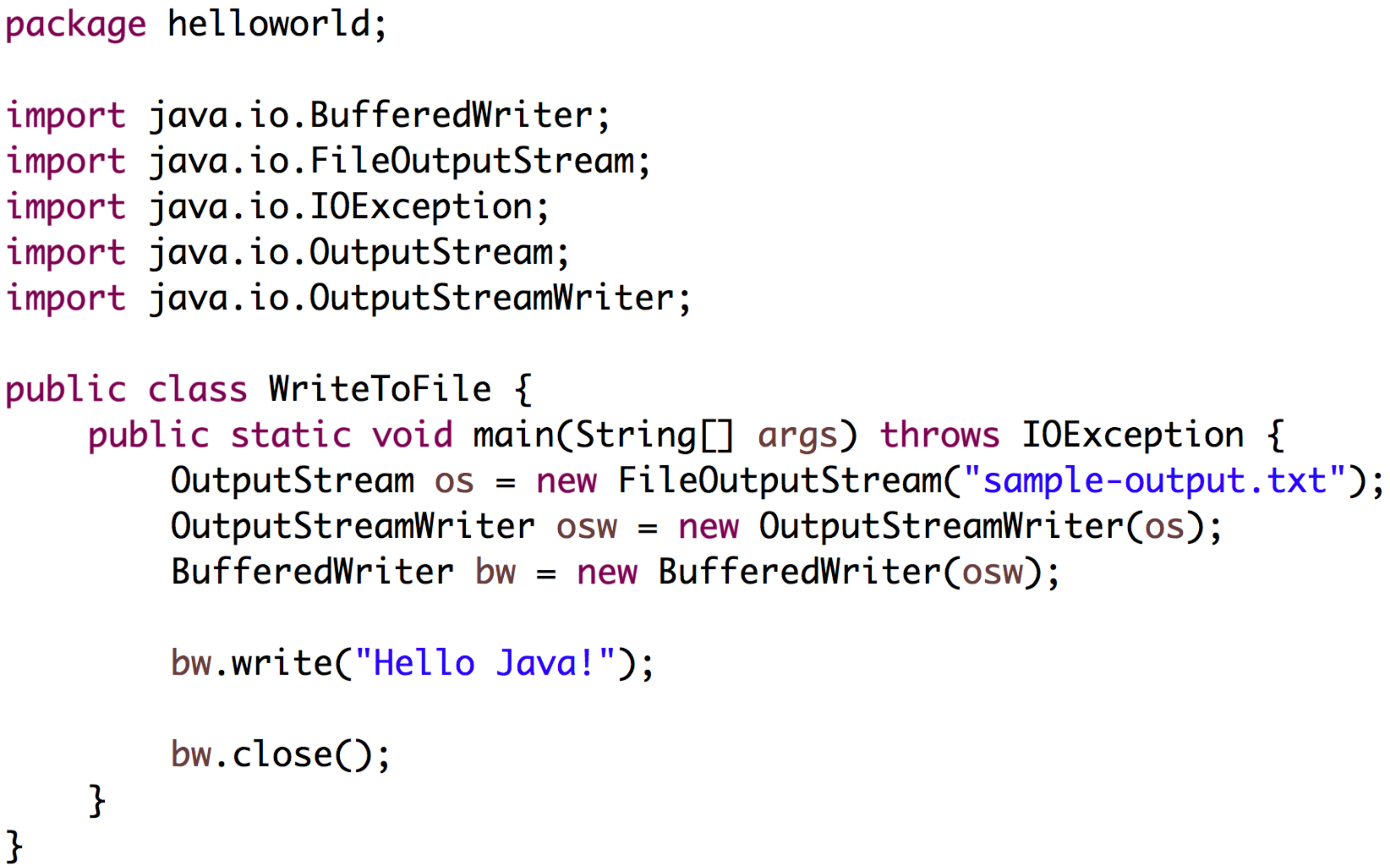


```java
package helloworld;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

public class ReadWholeFile {
    public static void main(String[] args) throws IOException {
        InputStream is = new FileInputStream("sample-input.txt");
        InputStreamReader isr = new InputStreamReader(is);
        BufferedReader br = new BufferedReader(isr);

        String line = null;
        while ((line = br.readLine()) != null) {
            System.out.println(line);
        }

        br.close();
    }
}
```

OutputStream

- ‘OutputStream’ is an abstraction of a stream of bytes to be written.
    - System.out → an output stream to console
    - FileOutputStream → an output stream to file
    - SocketOutputStream → an output stream to network
- Like ‘Reader’ classes, ‘Writer’ classes convert values to bytes stream.

Write a String to File



```java
package helloworld;

import java.io.BufferedWriter;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.io.OutputStreamWriter;

public class WriteToFile {
    public static void main(String[] args) throws IOException {
        OutputStream os = new FileOutputStream("sample-output.txt");
        OutputStreamWriter osw = new OutputStreamWriter(os);
        BufferedWriter bw = new BufferedWriter(osw);

        bw.write("Hello Java!");

        bw.close();
    }
}
```

Outline

- Variables
- Operators
- Conditionals
- Loops
- Methods
- Standard I/O
- **Arrays**
- Classes

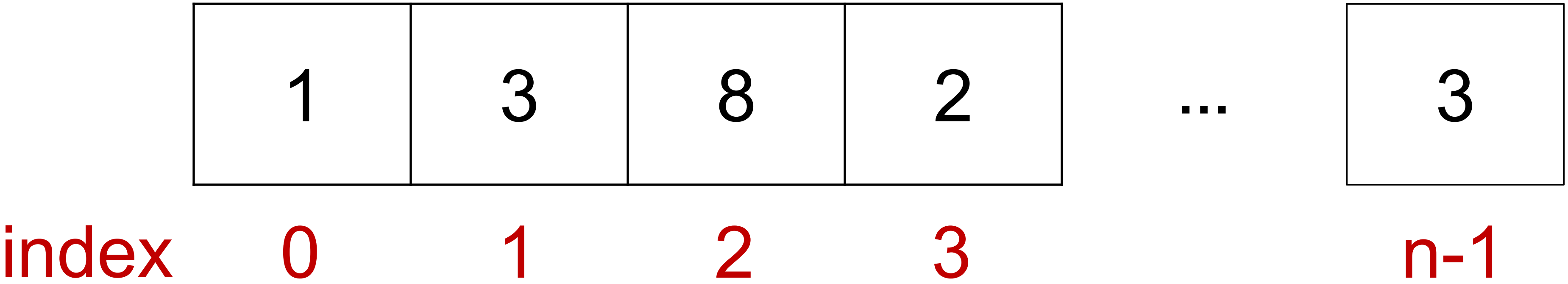
Arrays

- An array is a collection of data items that can be selected by indices.

- Items of an array can have any data type.
  - **`int`**, **`char`**, **`double`**, **`String`**, etc …

Arrays

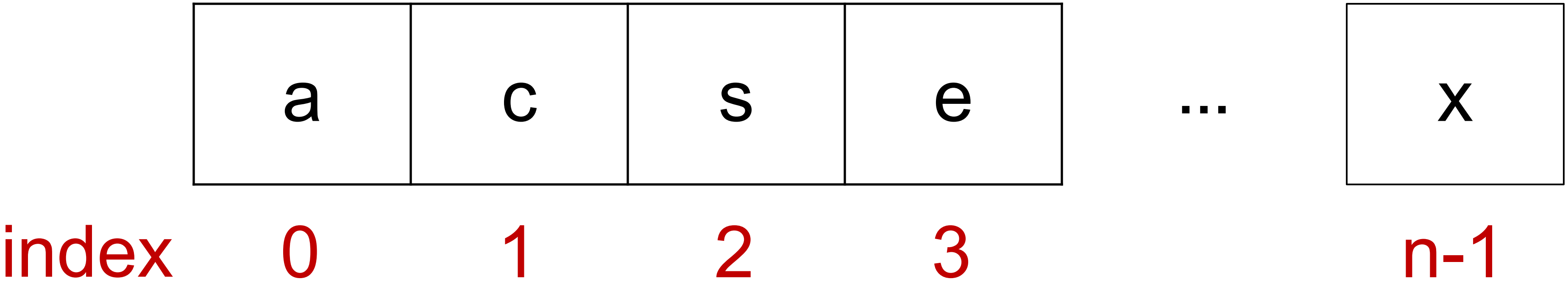
- Example 1
  - `int[]`



	1	3	8	2	…	3
index	0	1	2	3		n-1

Arrays

- Example 2
  - `char[]`

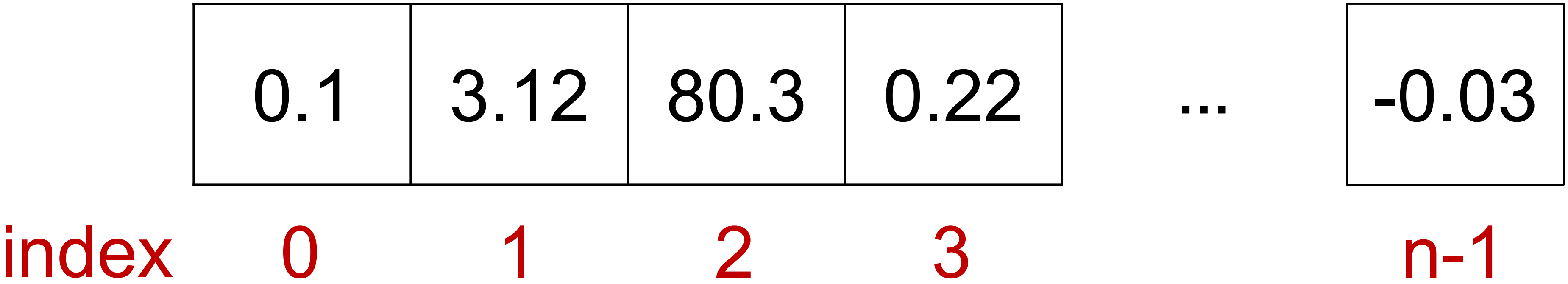


a c s e … x

index 0 1 2 3 n-1

Arrays

- Example 3
  - `double[]`



	0.1	3.12	80.3	0.22	...	-0.03
index	0	1	2	3		n-1

Arrays

- An array is defined by ‘TYPE[]’
- To create an array, use **`new`** operator
- Example 1

```
int[] example = new int[10];
```

- Example 2

```
int length = 10;
int[] example = new int[length];
```

Arrays

- The index of an array ranges from 0 to n-1.
  - n: the length of an array
- Example

```
int[] example = new int[10]; // declare an array
example[0] = 1; // assign 1 to the first element
example[1] = 2; // assign 2 to the second element
example[3] = 3; // assign 3 to the third element
example[9] = 4; // assign 4 to the fourth element
example[10] = 5; // Wrong
```

Arrays

- Initialize an array
  - Example 1

```
int[] example = {1,3,5,4,5};
```

1	3	5	4	5
0	1	2	3	4

Arrays

- Wrong examples
  - You can initialize an array only when you declare it.
    - Example 1

```
int[] example;
example = {1,3,5,4,5}; // Wrong
```

    - Example 2

```
int[] example = {1,3,5,4,5};
example = {1,2,3,4,5}; // Wrong
```

Arrays

- Wrong examples
  - All items of an array have a same type.
    - Example

```
int[] example = {1,3.5,5,4.2,5}; // Wrong
```

Arrays

- Access the elements of an array

```
array[index]
```

- Example 1

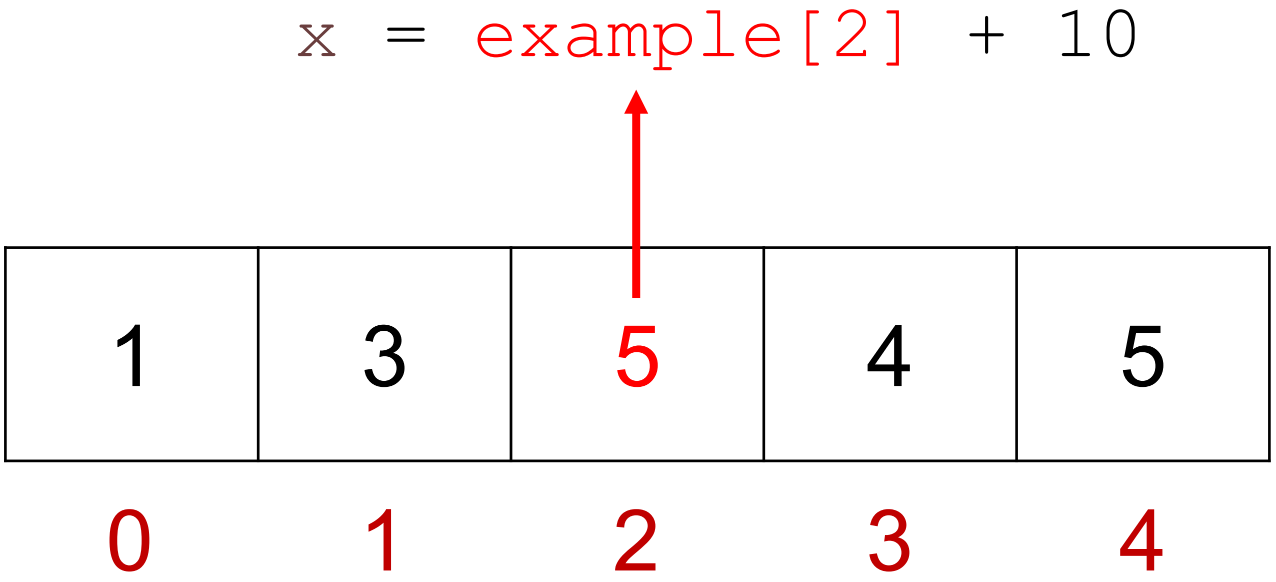
```
int[] example = {1,3,5,4,5};
example[3] = 10;
```

1	3	5	10	5
0	1	2	3	4

Arrays

- Example 2

```
int[] example = {1,3,5,4,5};
int x = example[2] + 10; // x = 5 + 10
```



```
x = example[2] + 10
```

1	3	5	4	5
0	1	2	3	4

Arrays

- An array has a `length` variable
- Example 1

```
int[] example = {1,3,5,4,5};
int length = example.length; // 5
```

- Example 2

```
int[] example = new int[10];
int length = example.length; // 10
```

Arrays

- You can create a multi-dimensional array.

```
int[][] example = new int[5][10];
```

- Access the elements of an array

```
array[index][index]
```

- Example

```
int[][] example = new int[5][10];
example[2][5] = 5;
```

Combining Arrays with Loops

- Typical usage of arrays are with loop statements.

```
int[] example = {1,2,3,4,5};
for(int i=0;i<example.length;i++){
     example[i] = example[i] * 2;
}
```

- You should be careful about the range of index variable (in this case, range of variable i).

Fibonacci: Revisited

- The code below shows calculation of Fibonacci number with an array instead of recursive function.

```
int N = 15;
int[] fibo = new int[N];
fibo[0] = 1;
fibo[1] = 1;
for(int i=2;i<N;i++){
    fibo[i] = fibo[i-1] + fibo[i-2];
}
```

Outline

- Variables
- Operators
- Conditionals
- Loops
- Methods
- Standard I/O
- Arrays
- **Classes**

Classes

- Example

```
public class Dog{
    String name;
    String species;
    int age = 0;
    double weight;

    void birthday(){
        age += 1; //a dog is an year older
    }
}
```

Declare a Class

- Declare a class

```
public class classname {
```

fields

methods

```
}
```

Declare a Class

- Fields

```
public class CLASSNAME {
      TYPE name;
      TYPE name = value;
}
```

- Example

```
public class Dog{
      String name;
      String species;
      int age = 0;
      double weight;
}
```

Declare a Class

- Declare a constructor
  - A constructor name is the same as the class name.
  - Constructors do not need a return type.

```
public class CLASSNAME {
      CLASSNAME (ARGUMENTS){


      }
}
```

- Class instance

```
CLASSNAME objname = new CLASSNAME(ARGUMENTS);
```

Declare a Class

- Example

```
        public class Dog{
                Dog(String _name, String _species,
double _weight){
                        name = _name;
                        species = _species;
                        weight = _weight;
                }
        }
```

Declare a Class

- Methods
- Example 1

```
public class Dog{
```

fields

```
      void birthday(){
            age += 1; //a dog is an year older
      }
}
```

Declare a Class

- Example 2

```
      public class Dog{
            void feed(double _food){
                  weight += _food;
                  System.out.println("The Dog gains 
weight.");
                  System.out.println("Current 
weight: "+weight);
            }
      }
```

Declare a Class

- Summarize a class declaration

```
public class Dog{
      String name;
      String species;
      int age = 0;
      double weight;

      void birthday(){…}
      void feed(double _food){…}
}
```

Using a Class

- Class instance

```
// class instances
Dog pet1 = new Dog("John", "Poodle", 2.5);
Dog pet2 = new Dog("Bob", "Beagle", 4.2);
```

Using a Class

- Access fields of instances

```
      CLASSNAME.FIELD
```

- Example

```
      System.out.println(pet1.name+"'s age is 
"+pet2.age);
      System.out.println(pet2.name+"'s 
species is "+pet2.species);
```

Using a Class

- Call methods of instances

```
CLASSNAME.METHOD(ARGUMENTS)
```

- Example

```
pet1.birthday();
pet2.birthday();
pet1.feed(0.3);
```

What You Need to Know

- Variables
  - Different types have different arithmetic
- Conditionals and loops
  - Remove repetitive parts in your programs
- Reading/Writing files
  - Most of programming assignments require this
- Dividing functional requirements into methods
- Array index starts with 0 (zero)
- Concepts of classes and instances

Questions?