Lecture 3: Loops, Arrays

Outline

- Conditionals
- Loops
- Arrays
- ArrayLists. Maps

if statement

if (***CONDITION***) {

    STATEMENTS

}

```
public static void test(int x) {
        if (x > 5) {
            System.out.println(x + " is > 5");
        }
}

public static void main(String[] arguments) {
        test(6);
        test(5);
        test(4);
}
```

Comparison operators

x > y: x is greater than y

x < y: x is less than y

x >= y: x is greater than or equal to x

x <= y: x is less than or equal to y

x == y: x equals y

(equality: ==, assignment: =)

Boolean operators

&&: logical AND

||: logical OR

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



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

else

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

```
public static void test(int x) {
    if (x > 5) {
        System.out.println(x + " is > 5");
    } else {
        System.out.println(x + " is not > 5");
    }
}

public static void main(String[] arguments) {
    test(6);
    test(5);
    test(4);
}
```

else if

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

```
public static void test(int x) {
    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");
    }
}

public static void main(String[] arguments) {
    test(6);
    test(5);
    test(4);
}
```

Outline

- Conditionals
- Loops
- Arrays
- ArrayLists. Maps

Loops

Loop operators allow to loop through a block of code.

There are several loop operators in Java.

The *while* operator

```
while (condition) {
    statements
}
```

The *while* operator

```
int i = 0;
while (i < 3) {
    System.out.println("Rule #" + i);
    i = i+1;
}
```

Count carefully

Make sure that your loop has a chance to finish.

The *for* operator

```
for (initialization;condition;update){
    statements
}
```

The *for* operator

```
for (int i = 0; i < 3; i=i+1) {
    System.out.println("Rule #" + i);
}
```

Note: `i = i+1` may be replaced by `i++`

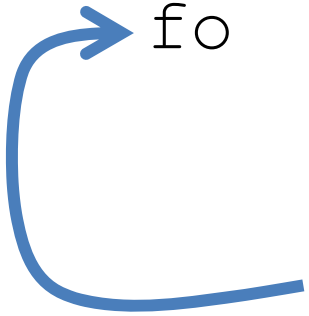
Branching Statements

break terminates a *for* or *while* loop

```
for (int i=0; i<100; i++) {
    if(i == 50)
        break;
     ystem.out.println("Rule #" + i);
```

Branching Statements

continue skips the current iteration of a loop and proceeds directly to the next iteration



```
fo   (int i=0; i<100; i++) {
     if(i == 50)
      continue;
     System.out.println("Rule #" + i);
}
```

Embedded loops

```
for (int i = 0; i < 3; i++) {
    for (int j = 2; j < 4; j++) {
        System.out.println (i + " " + j);
    }
}
```

Scope of the variable defined in the initialization: respective *for* block

Outline

- Conditionals
- Loops
- Arrays
- ArrayLists. Maps

Arrays

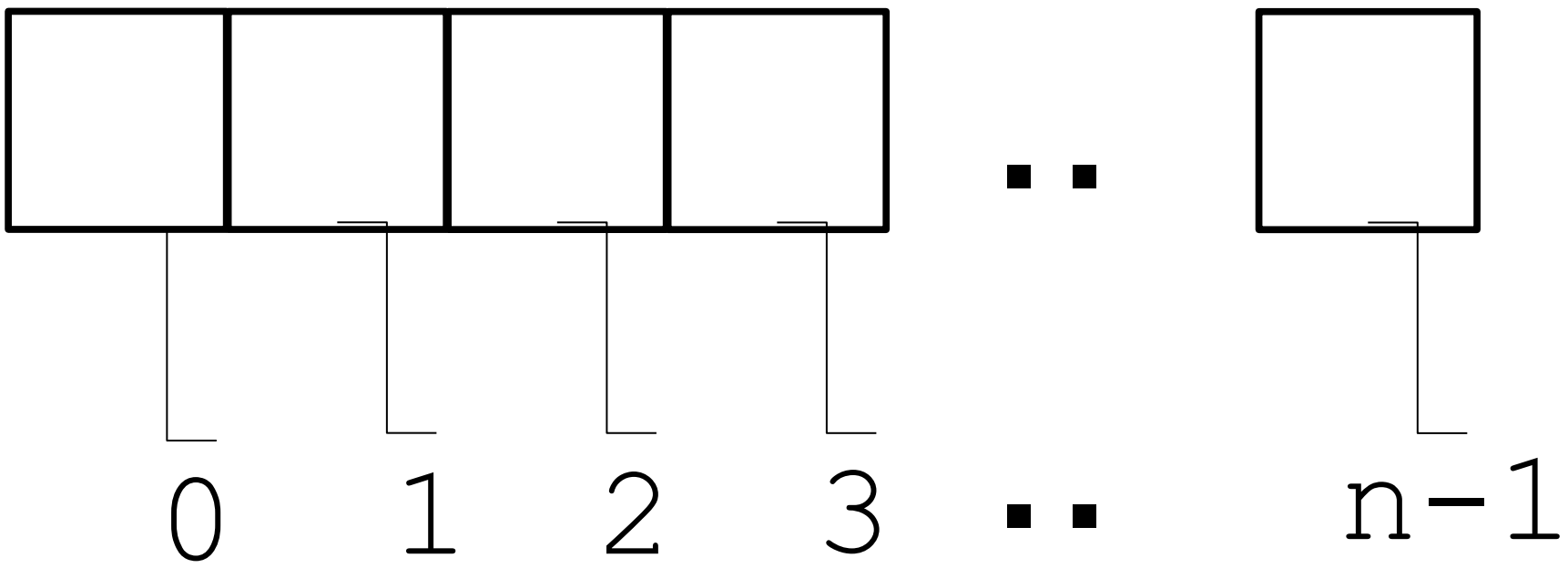
An array is an indexed list of values.

You can make an array of any type

int, double, String, etc..

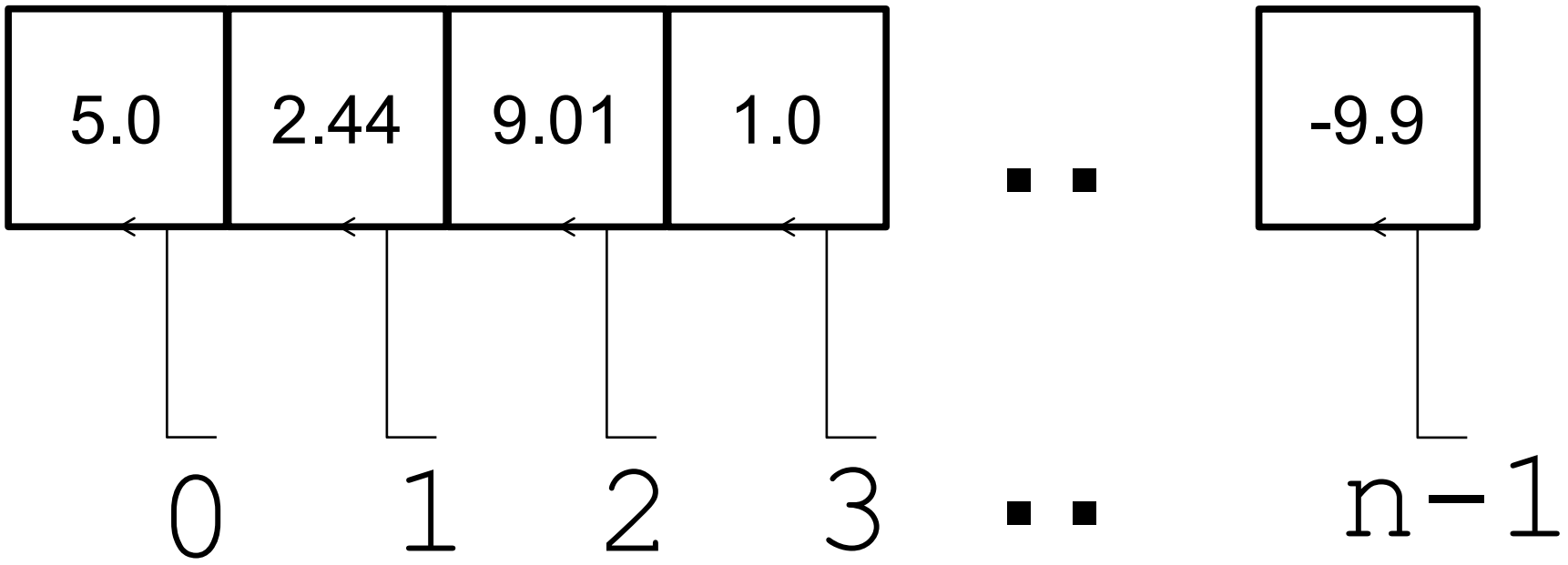
All elements of an array must have the same type.

Arrays



Arrays

Example: double [ ]



Arrays

The index starts at zero and ends at length-1.

Example:

```
int[] values = new int[5];
values[0] = 12; // CORRECT
values[4] = 12; // CORRECT
values[5] = 12; // WRONG!! compiles but
                // throws an Exception
                // at run-time
```

Have a demo with runtime exception

Arrays

An array is defined using TYPE[].

Arrays are just another type.

```
int[]   values;   // array of int

int[][] values;   // int[] is a type
```

Arrays

To create an array of a given size, use the operator `new` :

```
int[] values = new int[5];
```

or you may use a variable to specify the size:

```
int size = 12;
int[] values = new int[size];
```

Array Initialization

Curly braces can be used to initialize an array.

It can ONLY be used when you declare the variable.

```
int[] values  = { 12, 24, -23, 47 };
```

Quiz time!

Is there an error in this code?

```
int[] values  = {1, 2.5, 3, 3.5, 4};
```

Accessing Arrays

To access the elements of an array, use the `[]` operator:

```
values[index]
```

Example:

```
int[] values  = { 12, 24, -23, 47 };
values[3] = 18;              // {12,24,-23,18}
int x = values[1] + 3; // {12,24,-23,18}
```

The *length* variable

Each array has a `length` variable built-in that contains the length of the array.

```
int[] values =  new int[12];
int size =      values.length; // 12

int[] values2 = {1,2,3,4,5}
int size2 =     values2.length; // 5
```

String arrays

A side note

```
public static void main (String[] arguments){
    System.out.println(arguments.length);
    System.out.println(arguments[0]);
    System.out.println(arguments[1]);
}
```

Looping through an array

Example 1:

```
int[] values = new int[5];

for (int i=0;  i<values.length; i++) {
   values[i] = i;
   int y = values[i] * values[i];
   System.out.println(y);
}
```

Looping through an array

Example 2:

```
int[] values = new int[5];
int i = 0;
while (i < values.length) {
   values[i] = i;
   int y = values[i] * values[i];
   System.out.println(y);
   i++;
}
```

Outline

- Conditionals
- Loops
- Arrays
- ArrayLists. Maps

Arrays with items

Create the array bigger than you need

Track the next “available” slot

```
Book[] books = new Book[10];
int nextIndex = 0;

books[nextIndex] = b;
nextIndex = nextIndex + 1;
```

Arrays with items

Create the array bigger than you need

Track the next “available” slot

```
Book[] books = new Book[10];
int nextIndex = 0;

books[nextIndex] = b;
nextIndex = nextIndex + 1;
```

What if the library expands?

ArrayList

Modifiable list

Internally implemented with arrays

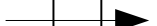
Features

- Get/put items by index
- Add items
- Delete items
- Loop over all items

Array → ArrayList

```
Book[] books =
      new Book[10];
int nextIndex = 0;


books[nextIndex] = b;
nextIndex += 1;
```



```
ArrayList<Book> books
= new ArrayList<Book>();


books.add(b);
```

```
import java.util.ArrayList;
class ArrayListExample {
    public static void main(String[] arguments) {
        ArrayList<String> strings = new ArrayList<String>();
        strings.add("Evan");
        strings.add("Eugene");
        strings.add("Adam");

        System.out.println(strings.size());
        System.out.println(strings.get(0));
        System.out.println(strings.get(1));

        strings.set(0, "Goodbye");
        strings.remove(1);
        for (int i = 0; i < strings.size(); i++) {
              System.out.println(strings.get(i));
         }
        for (String s : strings) {
            System.out.println(s);
        }
    }
}
```

Maps

Stores a (*key*, *value*) pair of objects

Look up the *key*, get back the *value*

Example: Address Book

- Map from names to email addresses

TreeMap: Sorted (lowest to highest)

HashMap: Unordered (pseudo-random)

```
public static void main(String[] arguments) {
    HashMap<String, String> strings = new HashMap<String, String>();
    strings.put("Evan", "email1@mit.edu");
    strings.put("Eugene", "email2@mit.edu");
    strings.put("Adam", "email3@mit.edu”);

    System.out.println(strings.size());
    strings.remove("Evan");
    System.out.println(strings.get("Eugene"));

    for (String s : strings.keySet()) {
        System.out.println(s);
    }
    for (String s : strings.values()) {
        System.out.println(s);
    }
    for (Map.Entry<String, String> pairs : strings.entrySet()) {
        System.out.println(pairs);
    }
}
```