IT ENTREPRENEURSHIP

Getting starting with JAVA

Slides by Taha Rushain

What is JAVA?

- **Programming Language**
- **Object Oriented**
- **High-Level Language**

Programming Language?

- **A human friendly language used to develop software programs that are understandable to computers**
- **Computers don't understand human language.**
- **There language only has 2 alphabets.**
- **We call there language, Binary.**
- **It's the language of 0s and 1s**

Object Oriented?

- **Before the concept of OOP, programs were perceived in the logical way only**
- **Input → Process → Output → End**
- **OOP brought the concept of Objects, just like in our physical world.**
- **Objects that have relationship. (It's Complicated!)**

Brief History

- **James Gosling, Mike Sheridan, and Patrick Naughton.**
- **Sun Microsystems lunched it.**
- **Later Oracle acquired it**

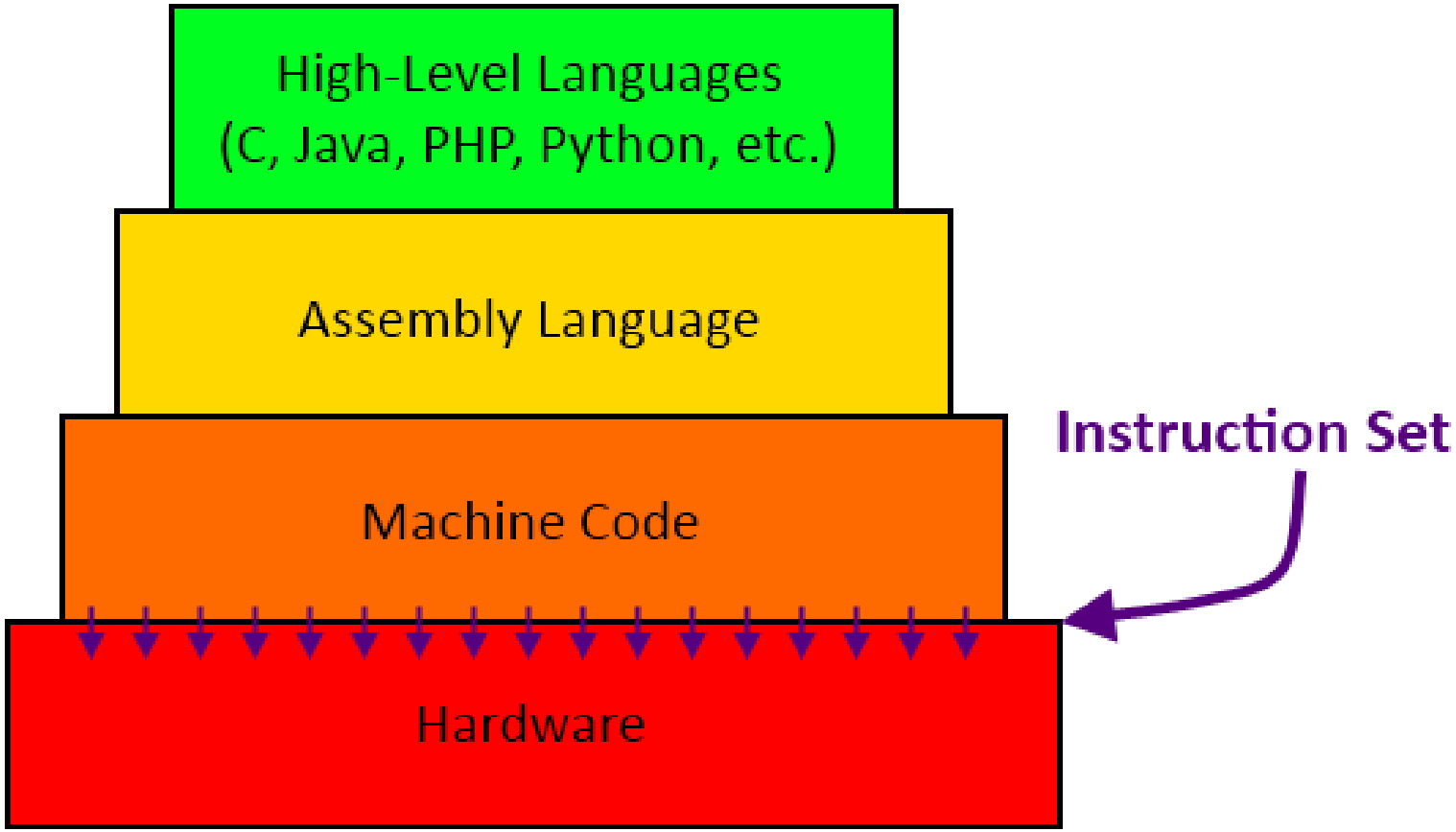
Write Once, Run Everywhere!

Buzz Lightyear Knows!

Is it Possible?

- **Yes, Java made this possible.**
- **But with some trade-offs.**
- **Java Virtual Machine.**

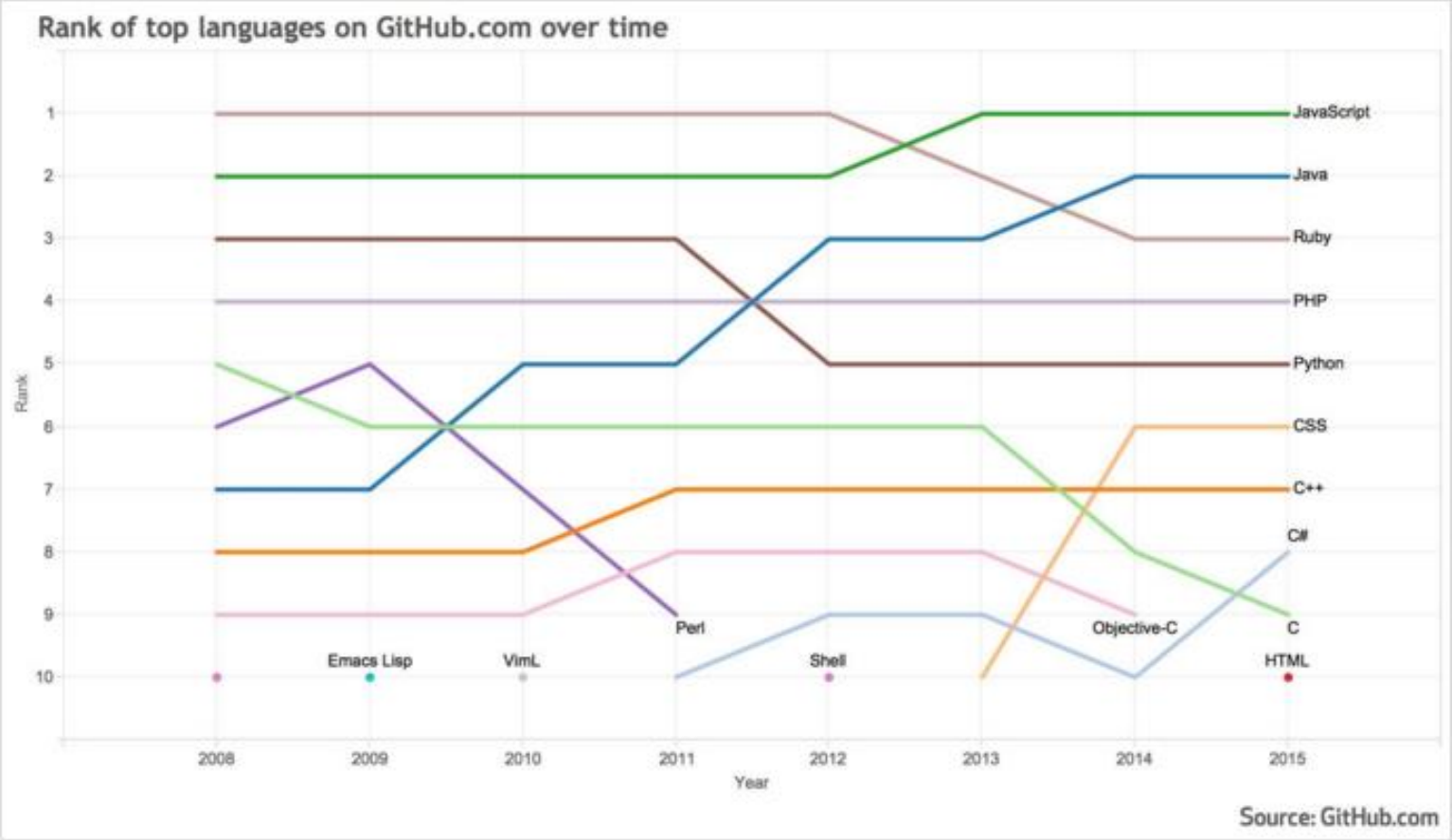
Layers of Human-2-Machine Communication



Is Java Worth it?

- **Every language has its own pros and cons.**
- **Different people prefer different languages based on different preferences.**
- **The choice is ultimately your!**

Cheers! Java is 2nd most popular



Whats Next?

- **Installation**
- **Then getting our hands dirty**

Getting Java

- Download and install java 8 JDK
- http://www.oracle.com/technetwork/java/javase/downloads/index.html

Now Eclipse

- Download eclipse
- http://www.eclipse.org/downloads/

CPU Instructions

- z = x + y
- Read location x
- Read location y
- Add
- Write to location z

Hello World

```
class Hello {
    public static void main(String[] arguments) {
        // Program execution begins here
        System.out.println("Hello world.");
    }
}
```

Basics

Second Program

```
class Hello2 {
    public static void main(String[] arguments) {
        System.out.println("Hello world.");  // Print once
        System.out.println("Line number 2");  // Again!
    }
}
```

Primitive Data types

- boolean: Truth value (true or false)
- int: 32-bit (4-byte)
- short: 16-bit (2-byte)
- float: 32-bit (4-byte) floating-point
- double: 64-bit (8-byte) floating-point
- char: 16-bit character

Variables

- Declare
- DATA_TYPE name;
- String foo;

Variable Assignment

- String foo;
- foo = “IAP 6.092”;
- String foo = “IAP 6.092”;

Third Program

```
class Hello3 {
    public static void main(String[] arguments) {
        String foo = "IAP 6.092";
        System.out.println(foo);
        foo = "Something else";
        System.out.println(foo);
    }
}
```

Operators

- Assignment: =
- Addition: +
- Subtraction: -
- Multiplication: *
- Division: /
- Modulus: %

Order of Operations

1. Parentheses
2. Multiplication and division
3. Addition and subtraction

Fourth Program

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

Fifth Program

```
class DoMath2 {
    public static void main(String[] arguments) {
        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);
    }
}
```

String Concatenation (+)

- String text = "hello" + " world";
- text = text + " number " + 5;

Order of Operations

- double x = 3 / 2 + 1;
- What would ‘x’ hold after this statement?
- double y = 3 / (2 + 1);

Mismatched Types

- String five = 5; // ERROR!

Type Casting

- int a = 2; // a = 2
- double a = 2; // a = 2.0 (Imp licit)
- int a = 18.7; // ERROR
- int a = (int)18.7; // a = 18
- double a = 2/3; // a = 0.0
- double a = (double)2/3; //0.666

Methods

Methods

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

Adding Methods

```
public static void NAME() {
    STATEMENTS
}
```

To call a method:

```
NAME();
```

Working with methods

```
class NewLine {
    public static void newLine() {
        System.out.println("");
    }

    public static void threeLines() {
        newLine(); newLine(); newLine();
    }

    public static void main(String[] arguments) {
        System.out.println("Line 1");
        threeLines();
        System.out.println("Line 2");
    }
}
```

Parameters

```
public static void NAME(TYPE NAME) {
    STATEMENTS
}
```

To call:

```
NAME (EXPRESSION) ;
```

Example

```
class Square {
    public static void printSquare(int x) {
        System.out.println(x*x);
    }

    public static void main(String[] arguments) {
        int value = 2;
        printSquare(value);
        printSquare(3);
        printSquare(value*2);
    }
}
```

What's wrong here?

```
class Square2 {
    public static void printSquare(int x) {
        System.out.println(x*x);
    }

    public static void main(String[] arguments) {
        printSquare("hello");
        printSquare(5.5);
    }
}
```

What's wrong here?

```
class Square3 {
    public static void printSquare(double x) {
        System.out.println(x*x);
    }

    public static void main(String[] arguments) {
        printSquare(5);
    }
}
```

Multiple Parameters

[…] ***NAME*(*TYPE NAME*, *TYPE NAME*)** {

STATEMENTS

}

To call:

```
NAME(arg1, arg2);
```

Example

```
class Multiply {
    public static void times (double a, double b) {
        System.out.println(a * b);
    }

    public static void main(String[] arguments) {
        times (2, 2);
        times (3, 4);
    }
}
```

Method Return Values

```
public static TYPE NAME() {
    STATEMENTS
    return EXPRESSION;
}
```

`void` means “no type”

Example

```
class Square3 {
    public static void printSquare(double x) {
        System.out.println(x*x);
    }

    public static void main(String[] arguments) {
        printSquare(5);
    }
}
```

Example

```
class Square4 {
    public static double square(double x) {
        return x*x;
    }

    public static void main(String[] arguments) {
        System.out.println(square(5));
        System.out.println(square(2));
    }
}
```

Variable Scope

- Variables live in the block ({}) where they are defined (scope)
- Method parameters are like defining a new variable in the method

Variable Scope

```
class SquareChange {
    public static void printSquare(int x) {
        System.out.println("printSquare x = " + x);
        x = x * x;
        System.out.println("printSquare x = " + x);
    }

    public static void main(String[] arguments) {
        int x = 5;
        System.out.println("main x = " + x);
        printSquare(x);
        System.out.println("main x = " + x);
    }
}
```

Variable Scope

```
class Scope {
    public static void main(String[] arguments) {
        int x = 5;
        if (x == 5) {
            int x = 6;
            int y = 72;
            System.out.println("x = " + x + " y = " + y);
        }
        System.out.println("x = " + x + " y = " + y);
    }
}
```

Methods: Building Blocks

- Big programs are built out of small methods
- Methods can be individually developed, tested and reused
- User of method does not need to know how it works
- In Computer Science, this is called “abstraction”

Mathematical Functions

- Math.sin(x)
- Math.cos(Math.PI / 2)
- Math.pow(2, 3)
- Math.log(Math.log(x + y))

Conditions

If statement

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

Example

```
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 statement

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

Example

```
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 Statements

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

Example

```
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);
}
```

Assignment: Foo Corporation

- Method to print pay based on base pay and hours worked
- Overtime: More than 40 hours, paid 1.5 times base pay
- Minimum Wage: $8.00/hour
- Maximum Work: 60 hours a week

Conversion by method

- String five = 5; // ERROR!
- String five = Integer.toString (5);
- String five = “” + 5; // five = “5”

- int foo = “18”; // ERROR!
- int foo = Integer.parseInt (“18”);

Good programming style

- String a1;
- int a2;
- double b; // BAD!!

- String firstName; // GOOD
- String lastName; // GOOD
- int temperature; // GOOD
- Use indentation, whitespace and don't duplicate test

Frequent Issues

Frequent Issues (I)

- The signature of the main method cannot be modified.

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

Frequent Issues (II)

- Return values: if you declare that the method is not void, then it has to return something!

```
public static int pay(double basePay, int hours) {
    if (basePay < 8.0)   return -1;
    else if (hours > 60) return -1;
    else {
        int salary = 0;
        ...
        return salary
    }
}
```

Frequent Issues (III)

```
public static int pay(double basePay, int hours) {
  int salary = 0;     // OK
  …
  int salary = 0;     // salary already defined!!
  …
  double salary = 0; //salary already defined!!
  …
}
```

```
class WeeklyPay {

public static void pay(double basePay, int hours) {

    if (basePay < 8.0) {
        System.out.println("You must be paid at least $8.00/hour");
    } else if (hours > 60) {
        System.out.println("You can't work more than 60 hours a week");
    } else {
        int overtimeHours = 0;
        if (hours > 40) {
            overtimeHours = hours - 40;
            hours = 40;
        }
        double pay = basePay * hours;
        pay += overtimeHours * basePay * 1.5;
        System.out.println("Pay this employee $" + pay);
    }
}

public static void main(String[] arguments) {
    pay(7.5, 35);
    pay(8.2, 47);
    pay(10.0, 73);
}
}
```

Good programming style

The goal of good style is to make your code more readable.

Rule #1: use good (meaningful) names

```
String a1;
int a2;
double b;                // BAD!!

String firstName;  // GOOD
String lastName;   // GOOD
int temperature;   // GOOD
```

Rule #2: Use indentation

```
public static void main (String[] arguments) {
    int x = 5;
    x = x * x;
    if (x > 20) {
        System.out.println(x + " is greater than 20.");
    }
    double y = 3.4;
}
```

Rule #3: Use whitespaces

- Put whitespaces in complex expressions

```
// BAD!!
double cel=fahr*42.0/(13.0-7.0);

// GOOD
double cel = fahr * 42.0 / (13.0 - 7.0);
```

Rule #3: Use whitespaces

- Put blank lines to improve readability

```
public static void main (String[] arguments) {

    int x = 5;
    x = x * x;

    if (x > 20) {
        System.out.println(x + " is > 20.");
    }

    double y = 3.4;
}
```

Rule #4: Do not duplicate tests

```
if (basePay < 8.0) {
            ...
} else if (hours > 60) {
            ...
} else if (basePay >= 8.0 && hours <= 60){
            ...
}
```

BAD

Rule #4: Do not duplicate tests

```
if (basePay < 8.0) {
            ...
} else if (hours > 60) {
            ...
} else {
            ...
}
```

Loops

Loops

- What if you want to do it for 200 Rules?

```
static void main (String[] arguments) {
    System.out.println("Rule #1");
    System.out.println("Rule #2");
    System.out.println("Rule #3");
}
```

The while operator

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

The while operator

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

The while operator

- 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);
}
```

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

Embedded loops

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

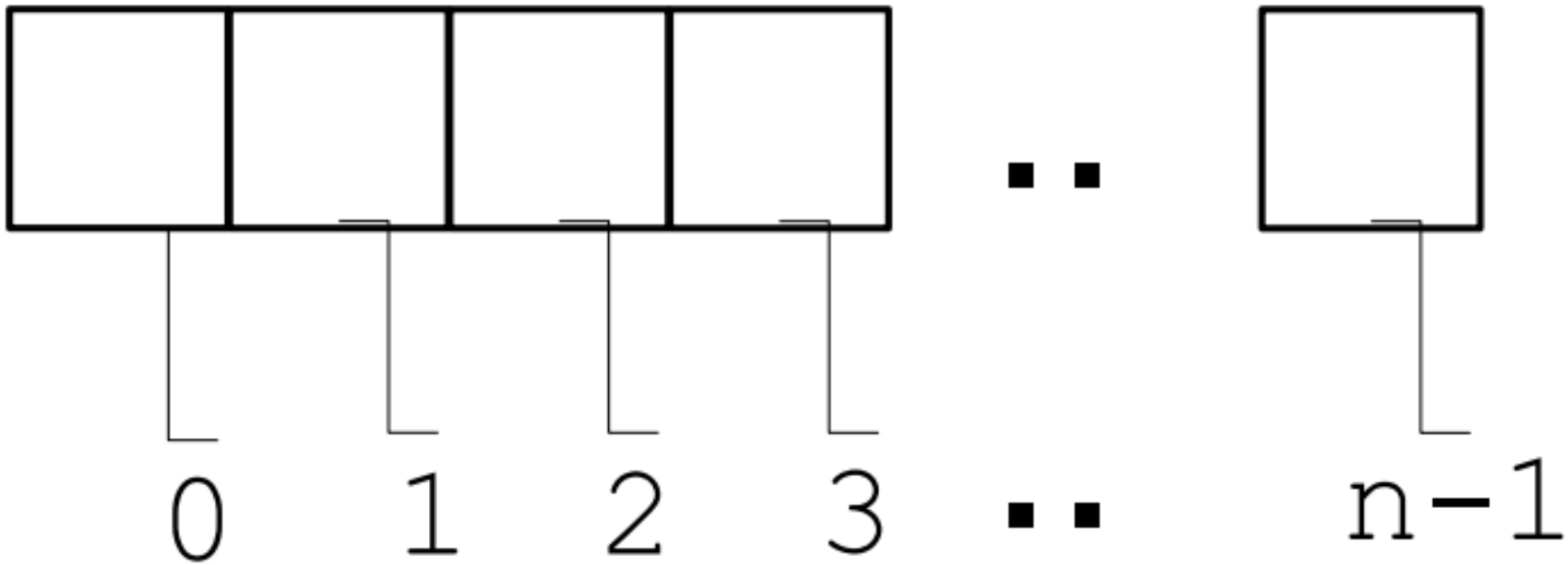
```
for (int i = 0; i < 3; i++) {
    for (int j = 2; j < 4; j++) {
        System.out.println (i + " " + j);
    }
}
```

Arrays

Arrays

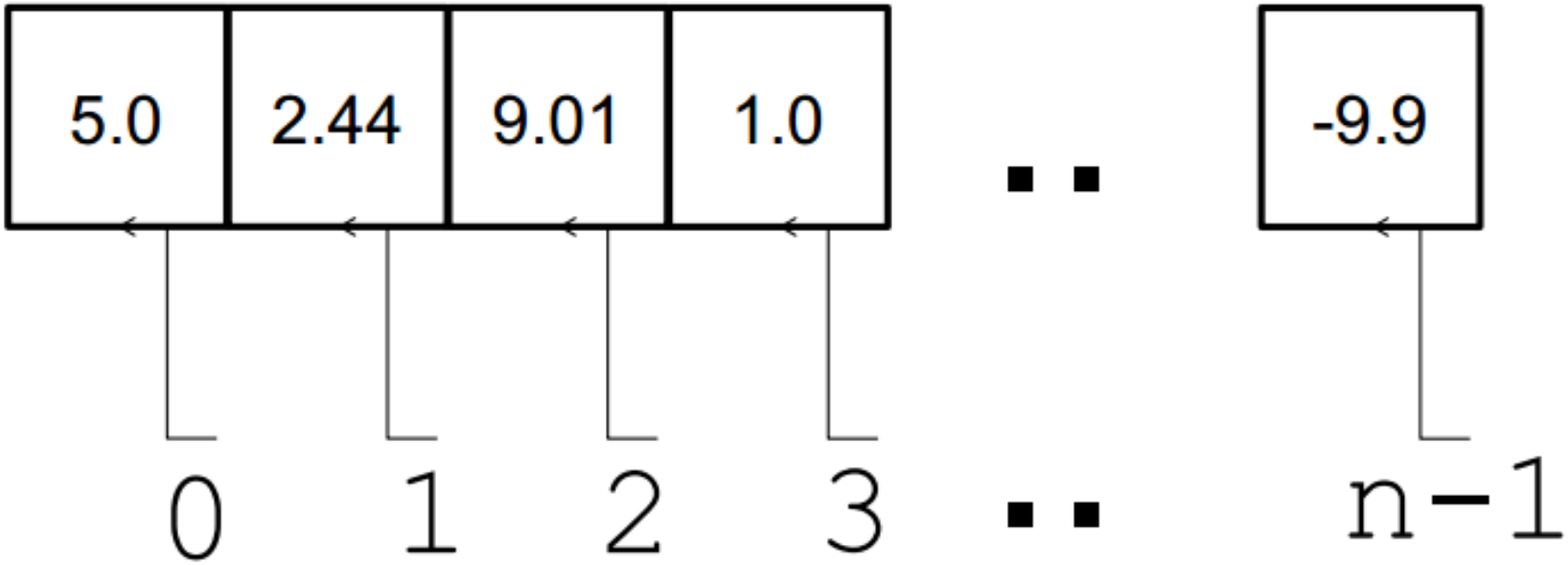
- An array is an indexed list of values.
- You can make an array of any type; int, double, String, etc..
- All elements of an array must have the same type. s

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.
- To create an array of a given size, use the operator new

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

Arrays

- 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
```

Combining Loops and Arrays

Looping through an array

```
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

```
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++;
}
```

Assignment 3

- A group of friends participate in the Boston Marathon.
- Find the best performer.
- Find the second-best performer.

Reference

- https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-092-introduction-to-programming-in-java-january-iap-2010/