Introduction to Java

https://tinyurl.com/y7bvpa9z

Eric Newhall - Laurence Meyers
Team 2849 Alumni

Java

Object-Oriented

Compiled

Garbage-Collected

WORA - Write Once, Run Anywhere

IDE

Integrated Development Environment

- Edit
- Build
- Debug

Hello World

```
class HelloWorld {
        public static void main (string[] args) {
                // Print “Hello World” to the console
                System.out.println(“Hello World!”);
        }
}
```

Braces mark code blocks

Statements end with semicolon

Comments are not compiled

Print

Write to console or log

Print vs Println

Variables

Store values

Strongly typed

Primitive Types

boolean (1-bit)
byte (8-bit integer)
short (16-bit integer)
int (32-bit integer)
long (64-bit integer)
float (32-bit decimal)
double (64-bit decimal)
char (16-bit unicode character)
String? (immutable object with special properties)

Objects

Objects are a combination of state (data), behavior (methods), and identity (unique existence among other objects)

Objects contain properties (other variables stored in the object)

Objects are created from a class constructor (using keyword new)

Objects may contain methods which modify values or perform other functions

Objects can be null

Constructors

```
class Person {
        int id;
        String name;
        int age;

        Person (int i) {
                id = i;
        }

        Person (int i, String n, int a) {
                id = i;
                name = n;
                age = a;
        }
}
```

Arrays

Collection/List of values of the same type

Length is defined when created

Accessed by index starting at 0

Operators

= (assignment)

Left side value equal to value of right side

+= -= *= /= %=

+ - * / %

Standard arithmetic operations

++ --

a++ || ++a || a = a + 1 || a +=1

Relational (returns boolean value)

- ==
- !=
- >
- >=
- <
- <=

Logical (return boolean value)

- &&
- ||
- !

Bitwise

- &
- |
- ^
- ~
- >>
- <<

Conditional ? :

- Value = <boolean> ? <true> : <false>;

Program Flow

```
if (<booleanA>){
        //code when booleanA is true
} else if (<booleanB>) {
        //code when booleanA is false and booleanB is true
} else {
        //code when booleanA and booleanB are false
}
```

```
switch (<variable>) {
        case <a>:
                //code to run when <variable> == <a>
                break;
        case <b>:
                //code to run when <variable> == <b>
                break;
        default:
                //code to run when <variable> not equal to any above values
                break;
}
```

Loops

```
while (<booleanA>){
        //code to loop while booleanA is true
        //check if booleanA is true before starting each loop
}

do {
        //code to loop while booleanA is true
        //check if booleanA is true at the end of each loop
} while (<booleanA>)
```

```
for (int i = 0; i < 10; i++) {
        //code to run 10 times
}

for (obj a : collectionOfObjs) {
        //code to run for each obj a in collectionOfObjs
}
```

Methods

Defined by a method header

```
public static void Main (string[] args)
```

And body

```
{
    //code in Main method
}
```

Public/Private/Protected

Public - All can access this method

Private - Method can only be accessed within the class in which it is defined

Protected - Method can only be accessed within the package in which it is defined

Return Types

All methods must have a return type which can be one of the following:

- Primitive - returns a primitive
- Object - returns an object
  - Constructors fall in this category
- Void - no return type

Parameters

Can have 0 to n

Can be of any type

Parameters are passed to the method by value

- Modifications to a parameter object will not persist outside of the method

Can overload a method by creating several of the same name with different parameters

```
public static void main( String[] args ) {
   Dog aDog = new Dog("Max");
   // we pass the object to foo
   foo(aDog);
   // aDog variable is still pointing to the "Max" dog when foo(...) returns
   aDog.getName().equals("Max"); // true, java passes by value
   aDog.getName().equals("Fifi"); // false
}

public static void foo(Dog d) {
   d.getName().equals("Max"); // true
   // change d inside of foo() to point to a new Dog instance "Fifi"
   d = new Dog("Fifi");
   d.getName().equals("Fifi"); // true
}
```

```
public static void main( String[] args ) {
   Dog aDog = new Dog("Max");
   foo(aDog);
   // when foo(...) returns, the name of the dog has been changed to "Fifi"
   aDog.getName().equals("Fifi"); // true
}

public static void foo(Dog d) {
   d.getName().equals("Max"); // true
   // this changes the name of d to be "Fifi"
   d.setName("Fifi");
}
```

Static Methods

Method called on the class, not the object

Use when the method does not depend on a single instance of an object

```
public static double Sin(double a)

public static void Main(string[] args)
```

Static methods can only read and write to Static variables

Errors

Syntax
- Prevents the program from compiling
- Spelling errors
- Missing closing }) or ;

Runtime
- Invalid operations
- Null access
- Array out of bounds

Logic
- Unexpected result, but no compilation errors

Try-Catch(-Finally)

Can be used for safety and debugging

- Some methods require a try catch

Can define multiple exception types to catch

Can use generic exception type to catch all errors

```
try{
        //code to run that may cause exceptions
} catch (ExceptionType|OtherExceptionType e){
        //code to handle exception of type ExceptionType or OtherExceptionType
} catch (Exception e){
        //code for generic exception
        //all previously unhandled exceptions will be caught here
} finally { //finally block is optional
        //code to run no matter what
        //this will even run if you return in the try/catch
}
```