Intermediate Java – Part 1

Yotam Avivi

2018

Rethink Possible®

Course Objectives

- Get Familiar with more advanced Java Topics by Programming a Complete SW from Scratch
- Be able to code on your own in Object Oriented Applications in Java
- Get Familiar with Proper SW Design Principles
- Unit Test Development / TDD
- REST API Development
- REST APIs Testing

Project Development – TicketWala (Cinema Ticket Order Application)

- We shall develop step by step a software for ordering movie tickets
- Interface can be either:
  - Textual (CLI)
  - Web (HTTP/REST)
- We shall learn to use automated test framework:
  - JUNIT
  - RestAssured*

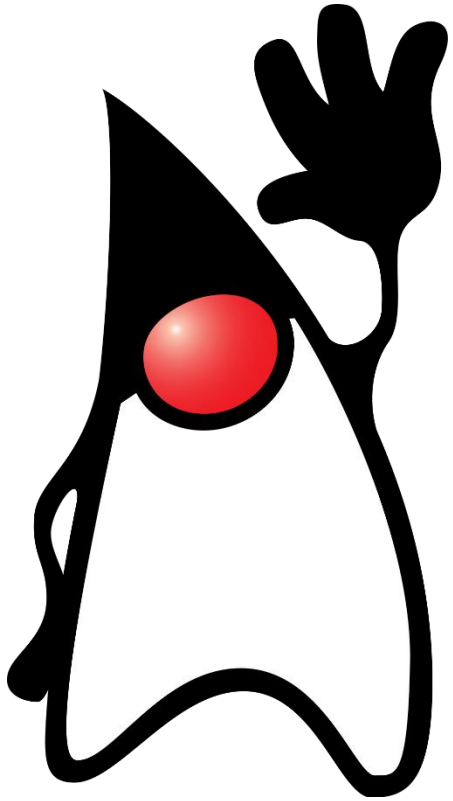
Java - Brief History

- Initiated in 1991 by James Gosling, Mike Sheridan, and Patrick Naughton
- Was originally designed for interactive TV (star7 PDA)
- Initially called Oak
- C/C++ Style Syntax
- JDK 1.0 Released in 1995 by Sun Microsystems (now Oracle)
- Last Release:

  Java SE 9 (Sep, 2017)
- ~ 9 Million Programmers

Java Language Aims to be…

- Simple
- Robust & Secure
- General Purpose
- Concurrent
- Object Oriented
- No Dependencies
- “Write Once, Run Anywhere”
- High Performance



Setup & Install Java & Java IDE (Eclipse)

Download JDK 8:

http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

Set JAVA_HOME

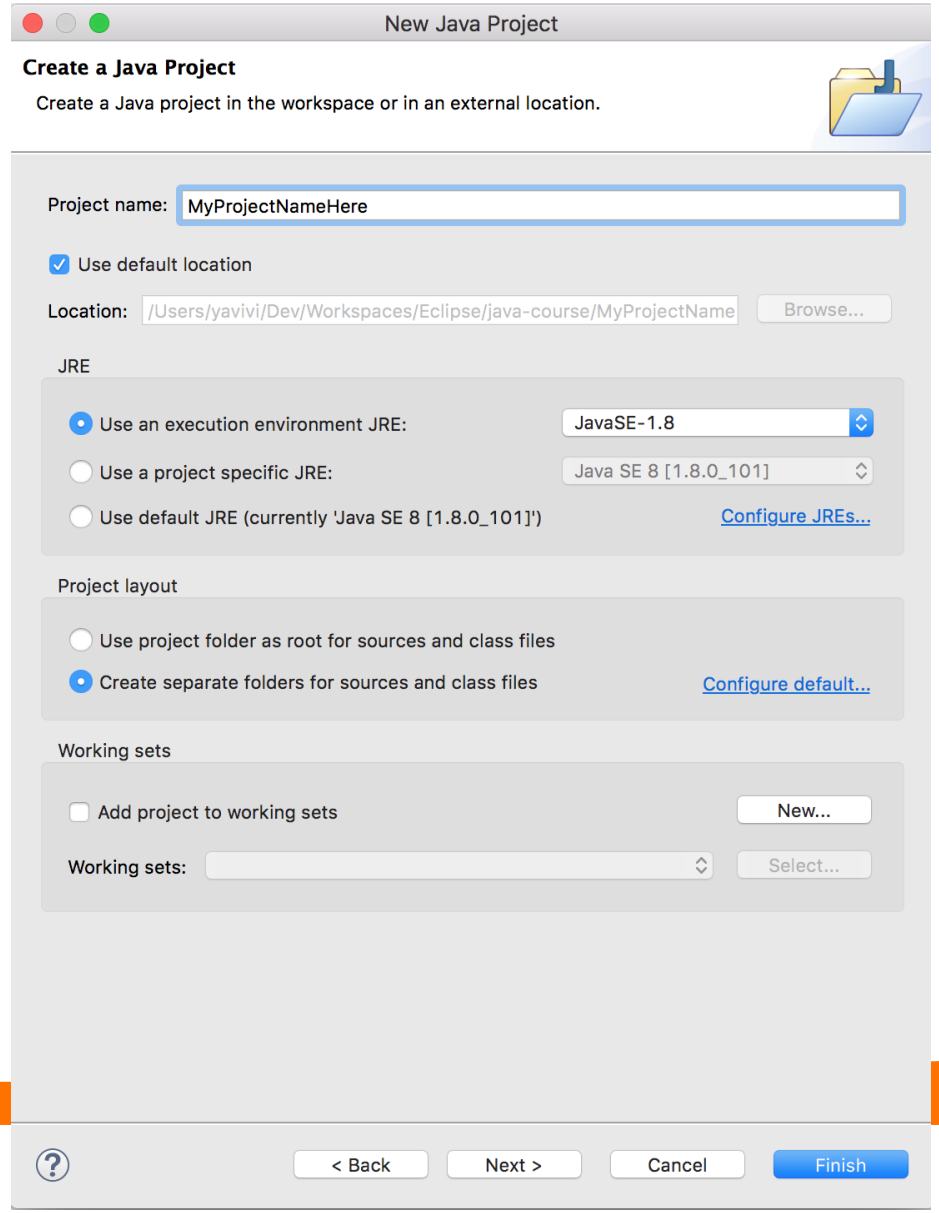
Set PATH

Download Eclipse (Eclipse for JEE Developers):

https://eclipse.org/downloads/

Intro to Eclipse IDE – Creating New Project

File → New → Java Project



Create New Class

File → New → Class

In order to run Java program **must** have a main method

Create Main Method

New Java Class

Java Class

Create a new Java class.

Source folder: java-course-2/src/main/java Browse...

Package: com.att.java.exercise

Enclosing type: Browse...

Name: Main

Modifiers: public package private protected

abstract final static

Superclass: java.lang.Object Browse...

Interfaces: Remove

Which method stubs would you like to create?

- [x] public static void main(String[] args)
- [ ] Constructors from superclass
- [x] Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

- [ ] Generate comments

< Back Next > Cancel Finish

Package – See next slides

Class Name

Parent Class.

All classes derived from Object by default

Java Packages

- A Java package is a technique for organizing Java classes into namespaces (appear as directories on FS)
- A package usually starts with organization domain and then its sub-domains separated by dots (each token is a sub-folder)
  E.g: *com.att.proj.controller*
- A good practice is to separate implementation classes and api classes to different packages
  E.g. *com.att.proj.service.api / com.att.proj.service.impl*
- Classes belong to same package should have high cohesion between them (relate to common task or job to be done)
- Different application components should communicate via APIs*

Java Packages – Some Examples

- java.io
- java.lang
- java.lang.annotation
- java.lang.instrument
- java.lang.invoke
- java.lang.management
- java.lang.ref
- java.lang.reflect
- java.math
  - BigDecimal.class
  - BigInteger.class
  - BitSieve.class
  - MathContext.class
  - MutableBigInteger.class
  - RoundingMode.class
  - SignedMutableBigInteger.class
- java.net
- java.nio

Java IO Package

Java Math Package

Java Networking Package

com.att.connectivity.api

- Client
- Server
- Message

com.att.connectivity.impl

- MyHttpServer
- MyTcpClient
- HttpMessage

API Classes

Implementation Classes

Public Static Void Main – Saying "hello" to StdOut

Class Modifier

Class Name

```
public class Main {
```

Return Value

Main Method

CLI Arguments

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


}
```

Method Modifiers

Prints to Stdout

Java Classes

A Class describes object's characteristics & capabilities

```
class Circle
private double radius
private String color
public double getRadius()
public double calcArea()
public double calcPerimeter()
```

A class has member variables

..and methods that can return values

return value

Access Modifiers controls visibility of methods and members. E.g.:

Public, private, protected

An object is an instance of a class with specific characteristics

Java Objects vs Primitives

- In Java, everything is either a primitive type or an Object
- Objects may hold various data while a primitive holds a single value
- Objects are always stored on the Heap (dynamic allocation memory area in Java) while primitives can also be stored on the stack
- The heap is created when the JVM starts up
- Heap may increase or decrease in size while the application runs
- When the heap becomes full, *garbage* is *collected*
- A primitive type is predefined by the language and is named by a reserved keyword (int, boolean, double etc.). See next slide.
- A Java program cannot define any other primitive data types.

Java Primitive Types

Type	Size	Values	Java Object
int	32-bit	-(2^31) – (2^31)-1	Integer
byte	8-bit	-128 - 127	Byte
long	64-bit	-(2^63) – (2^63)-1	Long
float	32-bit	-	Float
double	64-bit	-	Double
boolean	NA	true/false	Boolean
char	16-bit (Unicode)	0 - 65,535	Char

Declaring Classes

```
public class Circle {
        private double r; //Radius
        private String color;
        public Circle(double r, String c){
                this.r = r;
                this.color = c;
        }
        public double calcArea() {
                return Math.PI * r * r;
        }
}
```

Constructor – meant for creating the object

Notice the 'this' reserved word

Public Class Method that return Area of Circle as a number of type 'double'

Instantiating Objects from Classes

Reference to a new Circle Object in memory

A new Object is created and allocated in memory

```
Circle yellowCircle = new Circle(2.0, "YELLOW");

double area = yellowCircle.calcArea();

System.out.println("Circle area = " + area);
```

Invoke method and store value in a local var – 'area'.

Only public methods can be accessed from "outside world"

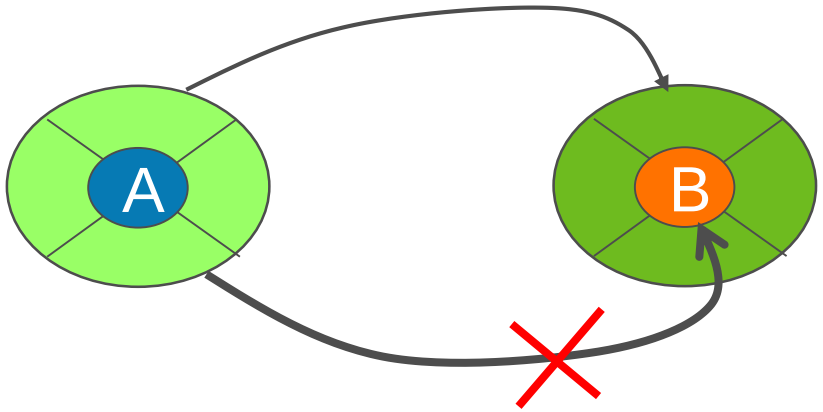
Print result to stout

String concatenation

The Three Principles of OOP

Encapsulation

Objects can hide their functions (methods) and data (instance variables) by declaring them as private or protected




```java
package com.att.java.exercise.oop;

public class A {

    public void moo() {
        B b = new B();

        //** Does not Compile **
        //** num is not accessible **
        //int n = b.num;

        //protected getNum() accessible because in same package
        int n = b.getNum();
        System.out.println("num=" + n);
    }

}
```

```java
package com.att.java.exercise.oop;

public class B {

    //Private - Accessible from within class only
    private int num = 5;

    //Protected - Accessible by: Class, Package, Subclass
    protected int getNum() {
        return num;
    }

}
```

Controlling Access using Modifiers

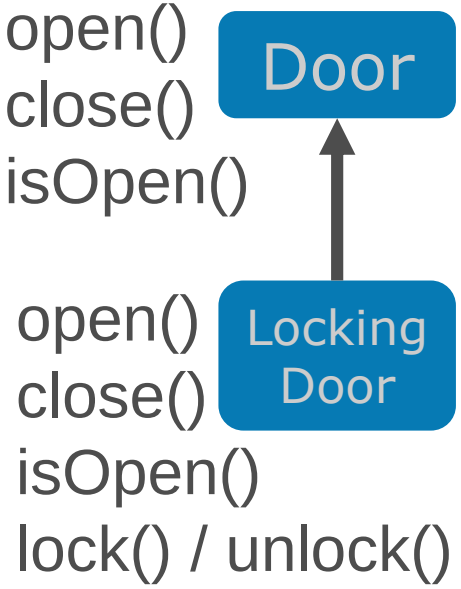
Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

The Three Principles of OOP (Cont.)

Inheritance

A subclass inherits all members of its superclass (variables & methods)

In Java every object inherits from Object class



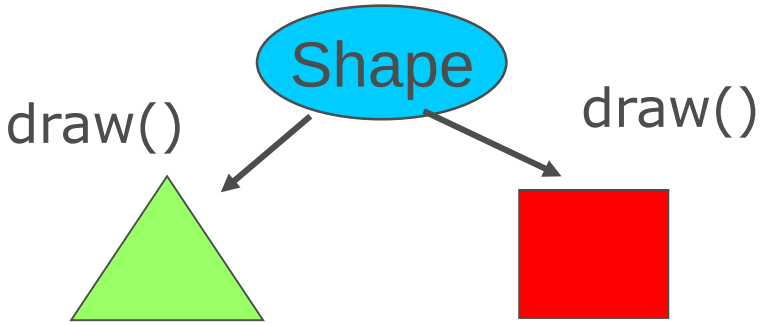

```java
public class Door {

    private boolean isOpen = false;

    public void open() {
        this.isOpen = true;
    }

    public void close() {
        this.isOpen = false;
    }

    public boolean isOpen() {
        return isOpen;
    }

}
```

```java
public class LockingDoor extends Door {

    protected boolean isLocked = true;

    @Override
    public void open() {
        if (isLocked) {
            System.out.println("Must unlock first!");
        } else {
            super.open();
        }
    }

    /**
     * Unlocks this door using provided key
     * @param key
     */
    public void unlock(Key key) {
        if (isValid(key)) {
            this.isLocked = false;
        } else {
            System.out.println("Must provide a valid key!");
        }
    }

     * Locks this door using provided key
    public void lock(Key key) {

    /**
     * Checks key
     * @param key
     * @return
     */
    private boolean isValid(Key key) {
}
```

The Three Principles of OOP (Cont.)

Polymorphism

Interface same despite different data types.
Shall invoke concrete class implementation




```
Shape shape1 = new Circle(5);
Shape shape2 = new Rectangle(5,10);

shape1.draw(); //Draws a Circle
shape2.draw(); //Draws a Rectangle
```

Inheritance – Exercise 1

1. Create Shape Class and Circle that extends it.

Shape should have:

- A (protected) color attribute
- A public calcArea(), calcPerimeter(), getColor(), setColor() methods (area and perimiter methods can either return -1 or throw an UnsupportedMethodException).

2. Create Circle Class that:

1. Extends Shape
2. Implements calcArea(), calcPerimeter()

* Read the next 2 slides to keep with coding conventions

Some Java Coding & Naming Conventions

Convention	Example
Package names should be lower case (domain name first)	com.att.ecomp.client.impl org.apache.http.server.api
Type names (Class/Interface etc.) are mixed case nouns starting with Upper Case	class MessageDispatcher class ScientificCalculator
Variable names should be Camel Case starting with lower case	numberOfAttempts userSelection
Constants should be upper case separated by underscore	MAX_ITERATIONS RED_COLOR
Method names should be verbs written in Camel Case	executeUserAction() openDvdPlayer()
get/set must be used where an attribute is accessed directly	rectangle.getWidth() rectangle.setColor(Color color)

Some Java Coding & Naming Conventions (Cont.)

Convention	Example
is prefix should be used for Boolean variables and methods	boolean isVisible() void setVisible(boolean isVisible)

Always use curly brackets in if statements and loops	if (a==1) {
	  b=2;
	}
	
	while (x<10) {
	  doSomething();
	}

Abstract Classes

- A class which is classified with the ‘abstract’ keyword
- Cannot be instantiated (cannot create new objects)
- May or may not have an abstract method
- Abstract method MUST be implemented by inherited (concrete) classes

Abstract Classes - Example

```
public abstract class Shape {
  . . .
  public Shape(Color c){
     this.color = c;
  }
  . . .

  public abstract double calcArea();
  public abstract double calcPerimiter();

}
```

Abstract Class Declaration
'**new** Shape'
is not allowed

Abstract Methods –
Must be implemented
in children

Inheritance & Abstract Classes – Exercise 2

1. Convert Shape class to Abstract class and its relevant methods.
2. Implement a Rectangle Class that extends Shape and implements

- abstract Shape methods (calcPerimeter, calcArea)
- Add an calcDiagonal method that calculates the Rectangle's diagonal length

3. Implement a Square Class

* Don't forget to keep with the coding conventions!!

OOP Operators: Casting / instanceof

```
Shape shape = createShape(); //A Shape object unknown in compile time

if (shape instanceof Square) {        //Determine Shape concrete type in RT

    Square square = (Square)shape;    //Down Casting the Shape to Square
    square.calcDiagonal();            //Safely invoking Square actions

}
```

Public Static Void Main (Cont.) – Reading from StdIn

```
public static void main(String[] args) throws IOException {
    System.out.println("Hi, what is your name?");
    Scanner scanner = new Scanner(System.in);
    String userName = scanner.nextLine();
    System.out.printf("Hello %s", userName);
    scanner.close();
}
```

Special Utility Object to read & parse text from console

Read next line into local var

Also available: nextInt(), nextDouble(), next Boolean() etc. to read other data types

C++ like printf

Release resources on shutdown

If - Conditional Statements

Boolean Statement (evaluated to “true” or “false”)

```
if (HOUR < 12) {
        sayGoodMorning();
} else {
        sayGoodAfterNoon();
}
```

Executed when “true”

Executed when “false”

<u>Can also be written as follows:</u>

```
HOUR < 12 ? sayGoodMorning() : sayGoodAfterNoon();
```

If - Conditional Statements Cont.

&& = AND Operator

|| = OR Operator

Second boolean statement is evaluated only if first one is true

```
if ( cpuUsage > 90 && cpuUsage <=100 ) {

        sendCriticalOverloadAlarm();

}
```

More Relational Operators: ==, !=, >, >=, <, <=

Reading from STDIN + Conditionals – Exercise 3

Display an arithmetic exercise of 2 numbers and ask the user for the result.

Print “Correct” / “Wrong” depends on the user answer.

String and String Operations

- String are an **immutable** (cannot be changed) sequence of Characters
- If you need **mutable** (changeable) Strings use StringBuffer object

Creating Strings Example:

```
String str1 = "AaBbCc";                    //Creates a String
char[] charSeq = {'A','a','B','b','C','c'};  //An Array of Chars
String str2 = new String(charSeq);         //Create a new String based on charSeq
String str3 = "Aa" + "Bb" + "Cc";          //Concatenating Strings
```

str1, str2, & str3 are equal (but they are not '==' to each other) – Why?

More String Ops (Cont.)

```
s.length();                          //Returns length of String

s.substring(beginIndex);             //Returns substring of string starting from beingIndex

s.trim();                            //Trims white spaces from edges

String[] tokens = s.split("\\s+"); //Splits string by spaces. Notice the regex.

String.format("First Name: %s, Last Name %s", fname, lname); //Formats this string
```