5 Day - JavaSE 6 Introduction

Stuart Jones

Overview

- 5 Days - Intro to Java Programming
- Schedule
    - 10am-6pm
    - Lunch at 1:30
- About Me
- About this Class
    - Java is a vast ocean
    - Learning how to find answers is key
- Each Java Topic
    - No Slides
    - Chat and Code Demo - Ask Questions!
    - Do - Pretend I'm the PdM
    - Review Volunteer Together
    - JUnit Tests where feasible
- Checkpoints
- Final Project (flexible)
- About the Book
- About You
- Questions?

About This Course

- Begin JavaSE 6 Development
- Compile, run and debug Java applications
- Understand the Object Oriented Concepts
- Use Eclipse IDE
- JDBC
- JUnit 4

Learning Objectives

- Get Set Up for JavaSE 6 Development
- Know how to compile, run and debug Java applications
- Know how to create an application from scratch
- Understand the following java topics:
    - Classes and Objects
    - Primitive Data Types
    - Arrays
    - Collections
    - Loop Structures
    - Returning Values
    - Passing Arguments
    - Branching Structures
    - Exception Handling
    - Basic File Access
    - Stream based IO
- Understand the following Object Oriented Concepts:
    - Abstraction
    - Encapsulation
    - Constructors
    - Static versus Instance
    - Inheritance
    - Polymorphism
- Understand how to use Eclipse IDE for Java
    - Code Short-cuts
    - Formatting Rules
- Understand How To use JDBC to:
    - Connect to a Database
    - Query a Database
- Understand How To use JUnit 4:
    - Test existing code
    - Test First with TDD
    - Code Coverage

Checkpoints

We stop and review, ask any questions you have on that topic and lets us ensure we know how to practically apply the concept / technique covered by the learning objective

Set up for Java Development

- Install the JavaSE (JDK 1.6 or 1.7)
    - Download
- Install Eclipse for Java Developers
    - Download
- Install Maven
    - Instructions
    - Download
- Install GIT
    - Download
- Verify java installation

```
$ java -version
```

```
java version "1.7.0_60"
Java(TM) SE Runtime Environment (build 1.7.0_60-b19)
Java HotSpot(TM) 64-Bit Server VM (build 24.60-b09, mixed mode)
```

- Verify maven installation

```
$ mvn -version
```

```
Apache Maven 3.2.1 (ea8b2b07643dbb1b84b6d16e1f08391b666bc1e9; 2014
Maven home: /usr/local/Cellar/maven/3.2.1/libexec
Java version: 1.6.0_65, vendor: Apple Inc.
Java home: /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Cont
Default locale: en_US, platform encoding: MacRoman
OS name: "mac os x", version: "10.9.3", arch: "x86_64", family: "m
```

- Verify GIT installation

```
$ git --version
git version 2.0.0
```

- Verify Eclipse Installation
    - Start Eclipse
    - Go to Preferences -> Java -> Installed JREs
    - Verify that the installed JRE is show in the list
- **CHECKPOINT:** Get Set Up for JavaSE 6 Development

Java Development Skills

- Explain the Java Development Workflow
    - `.java` File
    - `javac`
    - Compile-Time `CLASSPATH`
    - `.class` file
    - `java`
    - Run-Time `CLASSPATH`
    - Hello World!
-

- Write Hello World java application with no tools (Requires: text editor)

EXERCISE: Create projects folder

```
mkdir projects
cd projects
```

EXERCISE: Create a HelloWorld Java Application

```
cd projects
mkdir scratch
cd scratch/
mkdir src
vi src/HelloWorld.java
```

- `public static void` - Access Modifiers
- `main`
- `String[] args`
- `System.out.println()`
- Compile your application (Requires: javac from the JDK) (javac -verbose HelloWorld.java)

```
javac src/HelloWorld.java
```

- Run your Application (Requires: java from JRE)

```
java -cp ./src HelloWorld
```

- `-verbose` option:

```
javac -verbose src/HelloWorld.java
java -verbose -cp ./src HelloWorld
```

- `-d` option

- Intro CLASS_PATH

- **CHECKPOINT:** Know how to create an application from scratch

Java Platform Basics

- JDK - Java Development Kit (Tools)
- JRE - Java Runtime Environment (JVM + Class Libraries)
- JVM - Java Virtual Machine

Java Build Tools - Maven

- Generate a template application with Maven (App + AppTest)
    - `mvn archetype:generate`
- Use Maven to Build and Package a Java Application
- Use Maven to run JUnit Unit Tests for your code
- Import a maven project into Eclipse (Requires: Maven Integration for Eclipse)

Java Language Skills

Class Methods Intro

- Understand and use [Class Methods (static) esp. main]
    - No instance needed - We'll get to Instance Methods later
    - `String[] args`
    - `System.out.println()`
    - `public static void` - Access Modifiers

Packages

- Understand Java Packages ~ Namespace and Folder Name
- **EXERCISE:** Move your HelloWorld class to a package using eclipse

Data Types

- Understand and use Java Data Types

Local Variables

- Understand and use Local Variables
- Variable initialization rules

Printing Out Values

- System.out.printf
- Formatter
- Cheat Sheet

Exercise

- **EXERCISE:** Create a new folder called `datatypes` and a main application class called `DataTypeApp`. Put the `DataTypeApp` in an appropriate `package`. In DataTypeApp.main method, add a local variable of each of the built in data types and print out the values on separate lines, hint: `%n` translates to a new line

Checkpoint

- **CHECKPOINT:** Primitive Data Types

Expressions, Statements and Blocks

- Understand and use Expressions, Statements and Block
- Expression - evaluates to a value
- Statement - (expression, declaration or control flow)
- Blocks - a group of zero or more statements, hint at scope

Operators

Eclipse will show warnings and errors as you type, pay attention to them to learn how to use these operators

The Arithmetic Operators

- **EXERCISE:** Create a new folder called `operators` with a class called `OperatorsApp` in an appropriate package. For each of the Arithmetic operators `+,-,/,%,++,--` write an example usage and print out the results. Ensure that your app describes the difference between prefix and postfix `--/++`.

The Equality and Relational Operators

- **EXERCISE:** Create a new app any way you wish. In the main method, write an example expression for each of these operators which prints something out:
    - == equal to
    - != not equal to
    - > greater than
    - >= greater than or equal to
    - < less than
    - <= less than or equal to What data type do these expressions return?

Comments

- Understand and use Comments rest-of-line, in-line (and javadoc later)

Random Number Generator

This code is used in the next examples, we'll learn what all the pieces do later, for now just use this code in your exercises

```
Random random = new java.util.Random();
int randomIntBetween0and99 = random.nextInt(100);
/*
nextInt(int n)
Returns a pseudorandom, uniformly distributed int value between 0
*/
```

Control Flow

if statement

Simplest Control flow if-then

- if
- `else`
- `else if`
- Recommendation: Always use blocks

Exercise

- **EXERCISE:** Write an application which prints HEADS 50% of the time and TAILS 50% of the time to the console and exits

switch statement

- switch
- `break`
- `default`

Exercise

- **EXERCISE:** Write an application called `LuckyDay` which randomly selects a number between 0 and 6 (inclusive) and uses a `switch` statement to print out a day of the week in the format "This week, 'Wednesday' is your lucky day!". Where the day of the week printed is determined by the random number (Sunday is 0..Saturday is 6)

Checkpoint

- **CHECKPOINT:** Branching Structures

Arrays

Basic Arrays

- Understand Java Array Usage
- Allocate an array with the literal syntax
- Assign and access individual elements in the array

```
int[] anArray = {
  100, 200, 300,
  400, 500, 600,
  700, 800, 900, 1000
};
```

- `array.length`
- `System.arraycopy()`

Exercise

- **EXERCISE:** Re-write the `LuckyDay` application to use an array

Array Utilities

- Arrays

Command Line Arguments

Command Line Arguments (Java Language)

Enough hard coding! Let's read in some arguments

- `public static void main(String[] args)`
- How can we access args?

Exercise

- **EXERCISE:** Write a simple application in `sorter` which takes multiple command line arguments, puts them into an array, sorts the array and prints out the sorted array

Checkpoint

- **CHECKPOINT:** Arrays

Converting Values

- **DEMO:** Parsing Strings into primitive Types
- **EXERCISE:** Write a simple application in `cladder` which takes two command line arguments, checks that they are present, converts them into double values, multiplies them together and prints out the answer to two decimal places.

Loop Control Flow

while - loop

- while / do-while

Exercise

- **EXERCISE:** Write an application called `HighRoller` which utilizes one of these loop types to count the number of dice rolls required to reach `25`. Each dice roll is a random number between `1` and `6` inclusive. At the end, print out, the total score and how many rolls it took. If the player rolls exactly `25` then print out the special message `YOU ARE A HIGH ROLLER!!!`

for - loop

- for loop

```
for(int i=0; i < limit; i++)
{
}

for(;;)
{
  // Runs forever :) and this is a comment
}
```

Exercise

- **EXERCISE:** Write an application called `RandomArrayer`. The application should generate a random number between `10` and `20` (inclusive). Then allocate a new array of `int`s. The use a for loop to assign each `int` in the array to a random number. Use java.util.Arrays.sort() to sort the array of `int`s in ascending order and then another for loop to print out the numbers to the console

for each - loop

- for each

Exercise

- **EXERCISE:** Rewrite one of the loops in `RandomArrayer` to use the for each loop syntax

break - loop

- Shortcut exit loop

continue - loop

- Shortcut next iteration

Checkpoint

- **CHECKPOINT:** Loop Structures

Strings

String

- String is a special case object in that you can use a literal notation `"my string contents"` and get an object reference
- **DEMO:** Concatentation

Exercise

- **EXERCISE:** Review String concatentation, instance methods and try them out

StringBuilder

- StringBuilder

Exercise

- **EXERCISE:** Use StringBuilder to create a string by concatenating random values from an array

Methods

At the moment we are only *writing* `static` or Class Methods, we'll start writing instance methods later

Bu we are *calling* instance methods on the objects that we are working with.

Returning Values

- `return`
- void
- types

- Recommendation: Single Return Statement is preferred

- **DEMO:** Some different return values

- **EXERCISE:** Declare a method that returns any type without a body and See how eclipse helps you

- **CHECKPOINT:** Returning Values

Method Naming

- camelCase beginning with lowercase letter

Passing Parameters

- Understand argument passing and method invokation
    - Everything is passed by value, the value passed is either a primitive data type or an **object reference**
    - Objects themselves are always stored in the heap, and we only ever have a reference to that space
- **CHECKPOINT:** Passing Arguments

Class Methods

AKA `static`

- You don't need to create an instance to call `Class` (static) methods
- The instance of the `Class` is created when the bytecode is loaded by the `ClassLoader`
- `Math.min(a,b);`
- Our app is started when `main` is called
- Explore `Math`

Exercise

- **EXERCISE:** Use `mvn archetype:generate` and select `maven-archetype-quickstart` to generate and app template. In your `App` class use a loop and the Math library to print out a table with the radius, the area and the circumference of 10 circles with each radius between `1` and `10` units. Instead of printing the values out directly, write a `class` method called `getCircleInfo()` on your application class which takes an `array` of radii (type `double[n]`) and returns a 2 dimensional array `double[n][3]`, where in each row of the array the 0th element contains the radius, 1st the area, 2nd the circumference.
- **EXERCISE:** Replace the AppTest generated by the template with this AppTest Class to your Application. Try it out, by running the tests as a JUnit 4 Unit Test Case in eclipse. (The archetype template uses JUnit 3 syntax, so we are upgrading!)

```
package com.skillbox.boxes.circles;
// You'll need to change this package declaration, eclipse will he

import static org.junit.Assert.*;
import org.junit.Test;

public class AppTest {
    @Test
    public void testCirclrInfoResultLength() {
        double[] radii = { 1, 2, 3, 4 };
        assertEquals(radii.length, CircleApp.getCircleInfo(radii).
    }
}
```

Instance Methods

- If you have an Object Reference to an `instanceof` a `Class` you can call it's instance methods
- `String myString = "Contents of String";`
- `myString.contains(" of ");`
- Explore `String`
- How can you change a value of a `String`?
- **EXERCISE:** Use `mvn archetype:generate` and select `maven-archetype-quickstart` to generate and app template. In your `App` class write a class method called `reverseUpperCase` which takes a `String` parameter and uses some instance methods of the String class to return its contents in reversed UPPERCASE as a new `String`. Call your `reverseUpperCase` method 5 times from your main method with some different arguments including "" and print out the results
- **EXERCISE:** Add a Test!

TDD - Test Driven Development

- Red / Green / Refactor
- Write Tests before coding or refactoring

Looking Back at RandomArrayer (TDD)

Exercise

Use Test Driven Development to help you with these refactoring tasks:

- **EXERCISE:** Refactor the 'print out array' loop of `RandomArrayer` into a separate function - what would be a good return type/value?
- **EXERCISE:** Refactor the 'sorting' statement `RandomArrayer` into a separate function - what would be a good return type/value?

Self Study

Java Koans (General)

A Testy way to learn Java....

- Java Koans When you hit something we haven't covered we'll talk about it in class

Java Platform Differences

- Know the difference between JavaSE and JavaEE
- Java Platform Versions
- JavaSE 6 Overview
- We are only looking at a subset of JavaSE but you should be aware of:
    - JavaEE
    - JavaME vs Android API
    - JavaFX

Eclipse Skills

Eclipse Progjects

- Configure Project Settings
- Select a JRE

Debugging

- Run or Debug your code
- Set Debug Breakpoints
- View and edit runtime variable values
- Debug Your Tests

Checkpoint

- **CHECKPOINT:** Know how to compile, run and debug Java applications

Formatting

- Consistently Format Code
- Clean up!

Profiles

Eclipse Profiles

- Shared Java Formatting Profiles
- Always Format the same way to minimize conflicts

Checkpoint

- **CHECKPOINT:** Formatting Rules

Source Tools

Source Menu

- **DEMO:** Source Menu
    - Get/Setter etc.

Refactoring

- **DEMO:** Refactor Menu
    - Extract etc.

Code Coverage

- **DEMO:** Test Coverage
- Install Eclemma
- Run your tests with coverage as...

Checkpoint

- **CHECKPOINT:** Code Coverage

Eclipse Short Cuts

Keys

- **Cmd + 1** Fix any error!
- **Cmd + Opt + R** Rename in Place
- **CTRL + Space** code assist/code completion (happens automatically)
- **Cmd + Shift + F** Format Code
- **Cmd + Shift + O** Organize Imports
- **CTRL + /** Comment / Uncomment Toggle
- **CTRL + H** Uber Search
- **Cmd + Shift + T** Open Type
- **Cmd + Shift + P** Got to matching Brace
- **Cmd + [** Go to Previous Edit
- **Cmd +]** Go to Next Edit (after previous was used)

Looks useful:

- **Cmd + 3** Quick Access to...

Mouse Tips

- Select and Right Click....
- Double Click Break Point
- Mousefeed Plugin

Eclipse Features

- Eclipse -> Preferences -> Java -> Code Style -> Formatter
- Eclipse -> Preferences -> Java -> Code Style -> Clean Up
- Eclipse -> Preferences -> Java -> Editor -> Save Actions
- Source -> Generate XXXX
- Refactor -> Extract, Rename, Move

Other Resources

- A Cheat Sheet
- Versioned Cheat Sheets
- cs108
- Preferences
- 10 Day Tips

Numbers

Wrapper Classes

- For the built in numeric primitive data types we have wrapper types

Exercise

- **EXERCISE:** Investigate how to convert "1.61803398875" into a double value
- **EXERCISE:** What is AutoBoxing and Unboxing? Write a method which uses it.

Review

- Autoboxing

Object Oriented Principles

- Understand the following Object Oriented Concepts:
    - Inheritance
    - Abstraction and Encapsulation
    - Polymorphism

OO Overview

- Overview What Is an Object? What Is a Class? What Is Inheritance? What Is an Interface? What Is a Package? What Is Composition?

Abstraction, Encapsulation and Interfaces

When you turn the steering wheel, the front wheels turn, you don't care how it works

- Use StringBuilder as an example
- Abstraction (Interface / Observable Behaviour / Exposed)
- Encapsulation (Implementation / Hidden)
- 'The way you use it' is well defined
    - `interface` or `class` definition

- Clearly defined interfaces allow *Polymorphism*...

- **CHECKPOINT:** Abstraction

- **CHECKPOINT:** Encapsulation

Polymorphism

When you rent a car, it doesn't matter what model it is, you can still drive it

- Polymorphism
- Share some common nature but differ in implementation or additional behaviours
    - Interfaces with multiple implementations
    - `@Overrides` of instance methods
    - Implementations of `abstract` methods
- WikiPedia)

Inheritance and Classes

Not coding, just on paper

Exercise

- **EXERCISE:** Group discussion, select and brainstorm on domain
- **EXERCISE:** Now on your own in groups of two. Pick a domain and make some notes:
    - Describe Domain (package name)
    - Find Classes (`class`)
    - Find Inheritance (`extends`)
    - Draft up classes on paper
- **DISCUSSION:** Review of examples

Classes and Objects

If `Object` instances are Cookies then `Class`es are the cookie cutters

- Overview
- Class definitions define:
    - How instances must be created and initialized (Constructors, or other creation pattern)
    - What instances can do - behaviour - (their interfaces, instance methods)
    - How they do it - implementation
    - What data they store internally - fields
    - Who can access what
    - Any Class specific data, constants or behaviours (`static`)

Object Basics

- Every class extends `Object`
- `toString` -> type plus hashCode
- `getClass()` -> `Class`
- `instanceof`casting - A lot less of it now we have Generic Collections
- `clone` - `Cloneable` Marker interface

- `hashCode` and `equals` contract

- **LIVE DEMO:** Using `String` as an example

Class Basics

- `Class` extends `Object`
- `getName()` -> Useful for resources, jdbc drivers and loggers to avoid hard coding
- `getClassLoader()`

- `Class.forName()` -> `Class`

- **LIVE DEMO:** Using `String.getClass` as an example

Writing Java Classes

- Understand what these are:
    - Constructors, `new` and `super`
    - Static versus Instance methods and fields

Class Definitions

- One public class per .java file
- Must Match path with package
- Must Match Class name with Filename
- Constructors
    - `new`
    - `super`
    - `this`
- Fields
- Getters and Setters
- Review Class Methods and Class Variables
    - `static`
- Instance Methods and Instance Variables
    - non-`static`

Exercise

- **EXERCISE:** Implement the basics of your classes from the previous exercise.

Checkpoint

- **CHECKPOINT:** Constructors

Object Equality

- == vs `equals`
- `equals` and `hashCode`

Exercise

- **EXERCISE:** Write a Java class called `Employee` with fields for `firstName`, `lastName` and . Override `toString` in a meaningful way. Now try and implement equals and hashCode according to the documentation provided
- **EXERCISE:** Write some `@Tests` for your `equals` code

Demo

- **DEMO:** Eclipse Generators

Dates and Calendars (Java Platform)

- For this exercise `GregorianCalendar.html#GregorianCalendar(int, int, int)` might be handy

Exercise

- **EXERCISE:** Add `dateOfBirth` as `java.util.Date` to your Employee Class
- **DISCUSSION:** GregorianCalendar - Allows you to create dates and Calendar - Date Math
- **EXERCISE:** Run your tests again from the previous exercise

Access Modifiers

- Access Modifiers
- `public`
- `private`
- 'package' - The default

- `protected`

- **EXERCISE:** Discuss with your partner the appropriate access modifiers for parts of your class system

Inheritance Revisited

- Understand Class inheritance and how to implement it
    - Overriding
    - Abstract classes and methods
    - Final (classes, methods, variables, parameters)
- Recommendation: use `final` where you don't intend a value to change and avoid coding errors

Interfaces

- Understand Interfaces and how to use them
    - http://docs.oracle.com/javase/tutorial/java/IandI/index.html
- Interfaces are entirely abstract but can contain constants

Abstract Classes

- Understand Abstract Classes and how to define them

Demo

- **DEMO:** `Weapon` or `Shape` as `abstract`

Class Review

- Constructors Calling Constructors
- **CHECKPOINT:** Constructors
- Inheritance Review
- Method Overloading
- **CHECKPOINT:** Inheritance
- Static vs Instance Review
- **CHECKPOINT:** Static versus Instance
- **CHECKPOINT:** Classes and Objects

Collections Framework

The Java Collections Framework and Generics

- Collections Overview
- Generics - you may have noticed something new: `<E>`
- generics_collections.html

Collections Interfaces

- Overview
- List
- Map
- Set

Collections Implementations

There are many, we'll only cover the basics

- ArrayList
- HashMap
- HashSet

- **NOTE:** If you want to use your own classes as keys or in unique collections `hashCode` and `equals` will need to be implemented (later)

Exposing Collections

- `java.util.Collections`
- unmodifiableCollection
- `UnsupportedOperationException`

Collections Review

- Group discussion on collections

Exercise

- **EXERCISE:** Write a java method which generates an ArrayList of the Fibonacci series numbers starting with 0 and 1, up to and including the first value which exceeds `limit` where `limit` is a parameter. Recommendation use **TDD** and if the problem is unclear, ask your instructor to clarify

- **EXERCISE:** Where might we use HashMap? Implement an example

Checkpoint

- **CHECKPOINT:** Collections
- **CHECKPOINT:** Polymorphism

Generics

You saw something new in Collections ...

- Generics
- generics_collections.html
- Type Erasure - Generic information doesn't exist at runtime

Exceptions

Understand Java Exceptions

Discussion

Declaring Exceptions

- Checked vs Unchecked

Handling Exceptions

- How to handle Exceptions

Try - Catch - Finally

- finally - always gets done
- **DEMO:** `try`, `catch`, `finally`

Throwing Exceptions

- Help keep code simple and read-able
- Understand How and When to Throw exceptions
- When to:
    - Invalid Parameters
    - unexpected conditions
    - things outside of your control
- When not:
    - Not for control flow
- Throwing Exceptions

Exceptions Review

Exercise

- **EXERCISE:** Catch Exception - Create and catch an `ArrayIndexOutOfBoundsException.html`
- **EXERCISE:** Throw Exception - Example Check for null and throw a NullPointerException, document it

Checkpoint

- **CHECKPOINT:** Exception Handling

File Access

File (JavaSE6 Platform)

- File
- Reading and Writing Files is done using the Stream IO APIs

Exercise

- **EXERCISE:** Check File exists, list files in directory, check permissions, delete file

Stream Based IO

Streams (Java Platform)

- (NOT TO BE CONFUSED WITH Java8 Streams for Functional Programming)
- Streams IO
    - byte streams
    - character streams
    - Blocking
    - Use `try` and `finally` to close streams when done
    - Notices that the streams are declared and initialized to `null` outside of the `try` and `null` is tested in the `finally` block

Canonical Example: File Byte Stream (Java Platform)

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class CopyBytes {
    public static void main(String[] args) throws IOException {

        FileInputStream in = null;
        FileOutputStream out = null;

        try {
            in = new FileInputStream("source.jpg");
            out = new FileOutputStream("target.jpg");
            int c;

            while ((c = in.read()) != -1) {
                out.write(c);
            }
        } finally {
            if (in != null) {
                in.close();
            }
            if (out != null) {
                out.close();
            }
        }
    }
}
```

- **EXERCISE:** Implement a byte based file copy application which takes two paramters on the command line

Canonical Example: File Character Stream (Java Platform)

```
import java.io.*;

public class CopyFile {
   public static void main(String args[]) throws IOException
   {
      FileReader in = null;
      FileWriter out = null;

      try {
         in = new FileReader("source.txt");
         out = new FileWriter("target.txt");

         int c;
         while ((c = in.read()) != -1) {
            out.write(c);
         }
      }finally {
         if (in != null) {
            in.close();
         }
         if (out != null) {
            out.close();
         }
      }
   }
}
```

- **EXERCISE:** Implement a character based file copy application which takes two paramters on the command line

BufferedStreams

- bytes: `BufferedInputStream` and `BufferedOutputStream`
- character: `BufferedReader` and `BufferedWriter`
- `flush`

- **EXERCISE:** Implement a character based file copy application which takes two paramters on the command line using `BufferedReader` and `BufferedWriter`

Appending to a File

- `new FileWriter("filename", true)`
- **EXERCISE:** Use your file copy application as a basis for new utility function which takes a text file and converts all the uppercase letters to lowercase and vice versa. All other characters remain unchanged.
- Recommendation: write a utility method which applies the transformation to a given string first, and test it using TestCases. Then use that method in your app.
- @see `Character.isUpperCase()`
- **CHECKPOINT:** Basic File Access
- **CHECKPOINT:** Stream based IO

Sorting and Searching

Natural Ordering

- Natural Ordering

Comparisons

- Comparator vs Comparable
- **EXERCISE:** Implement Comparable with your `Employee` class
- **EXERCISE:** Put some `Employees` in an `ArrayList` and sort them

Searching

- Requires a sorted array
- Arrays Search
- **EXERCISE:** Sort and Array of `Employees` by date of birth using a comparator

Variable Scope

What is Scope?

- Understand variable scope and shadowing - com.skillbox.boxes.scope.ScopeExample

Variable Shadowing

- mXXX, sXXX or other naming conventions can assist in avoiding this

Exercise

- **EXERCISE:** Write a sample class that demonstrates variable shadowing and then make a promise to yourself not to do it again!
- **QUESTION:** What is the scope of the variables declared in a for loop?

Coding Generics

Generics In your Classes (Java Platform)

Exercise

- **EXERCISE:** Given this Interface:

Save it as *GenericStack.java*

```
interface GenericStack<T> {
  void push(T obj) throws GenericStackFullException;
  T pop() throws GenericStackEmptyException;
}
```

In separate files define `GenericStackEmptyException`, `GenericStackFullException` and a working implementation of `GenericStack` called `ArrayBasedGenericStack`.

`ArrayBasedGenericStack` should have a single constructor which takes an `int` parameter which is the size of the fixed array to be used to back up the Stack. We are looking for a *LIFO* : *Last in First Out* stack.

Before you start coding your implementation, write a JUnit 4 TestCase called `ArrayBasedGenericStackTest` which Test the behaviour. Use `@Test(expected = GenericStackEmptyException.class)` to annotate some of your tests

- Bounded and Unbounded Generics

Enums

Coding and Using Enums

- Java Enum Docs
- `.values() for each`
- `switch`
- `toString`
- **EXERCISE:** Recode the LuckyDay app using a `DayOfWeek` Enum and the `values()` array

Packages

Packages and Imports

File paths match package names

- packages and imports
- static imports

Developer Skills

Advanced Debugging

- Debug to a *remote*(local) application using Java Platform Debugger Architecture (JPDA) from Eclipse
    - agentlib options
- **EXERCISE:** Start the LuckyDay app with the debugger enabled and connect to it using Eclipse

Testing with JUnit4

JUnit4

- Know how to write JUnit test cases to Unit Test Your Code
- http://www.mkyong.com/tutorials/junit-tutorials/
- Pick a Mock Framework
    - Mockito
- **CHECKPOINT:** Test existing code
- **CHECKPOINT:** Test First with TDD

Maven By Example

An excellent tutorial

Let's work through the following sections:

3. A Simple Maven Project

and

4. Customizing a Maven Project

and

Convert the Unit Tests to JUnit 4 format

- Change the JUnit Dependency Version
- Google: `Maven JUnit Dependency`
- http://mvnrepository.com/artifact/junit/junit
- Pick the latest version 4 dependency
- Update the `pom.xml`
- Edit the tests to use `@Test` annotations

JDBC

If you have an existing database, research which JDBC driver you should use and the best way to connect. Hopefully there will be a DataSource available, otherwise you can use the JDBC Driver directly through the DriverManager

JDBC

JDBC Connections

- How to connect your application to a database with JDBC Connection
- Load / auto register the driver and then request a connection from the DriverManager with a valid URL
- Tutorial
- close `Connection` when done (if you opened it)

JDBC Review

Exercise

- **EXERCISE:**
- A project which reads a CSV File from disk and imports some of the data into a database
- DataSource
- When the file is imported a second time existing records are updated and new records are added
- OpenCSV
- Via Maven
- Get it working without batching first and then add batch queries
- **CHECKPOINT:** Connect to a Database
- **CHECKPOINT:** Query a Database

JDBC DataSources

- Application often run in containers which have configured `DataSource`s so this is most commmon, but you can create your own:
- How to connect your application to a database via a `DataSource`

Example from **boxes/checkout**

```
final SQLiteConfig config = new SQLiteConfig();
final SQLiteDataSource ds = new SQLiteDataSource(config);
ds.setUrl("jdbc:sqlite:" + props.getProperty("dburl", "db/checkout

// now use the ds to get a connection
```

Executing SQL

- Know how to execute SQL statements:
    - CREATE TABLE, SELECT, INSERT, UPDATE, DELETE, DROP TABLE

```
// Declare outside of try so we can close it in finally
Statement stmt = null;
String query = "SELECT ID, NAME FROM EMPLOYEE";

try {
    stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery(query);
    while (rs.next()) {
        int id = rs.getInt("ID");
        String name = rs.getString("NAME");
        System.out.println(nae + " : " + id);
    }
} catch (SQLException e ) {
    e.printStackTrace();
} finally {
    if (stmt != null) { stmt.close(); }
}
```

ResultSets

```
resultSet.getXXX();
```

- Retrieving Values

Executing SQL Updates

`executeUpdate`

Prepared Statements

```
  void update(final Connection conn, final String tableName)
      throws SQLException {
    final PreparedStatement prepStmt = conn.prepareStatement("UPDA
        + tableName + " SET NAME = ? WHERE ID = ?");
    prepStmt.setString(1, getProductName());
    prepStmt.setLong(2, getId());
    prepStmt.executeUpdate();
    prepStmt.close();
  }
```

Retrieving Generated Keys

- Getting IDs back

``` void insert(final Connection conn, final String tableName) throws SQLException { PreparedStatement prepStmt = null; ResultSet generatedKeys = null;

```
try {

  prepStmt = conn.prepareStatement("INSERT INTO " + tableName + SQ
      Statement.RETURN_GENERATED_KEYS);
  prepStmt.setString(1, getProductName());
  prepStmt.setInt(2, getStockLevel());

  final int affectedRows = prepStmt.executeUpdate();
  if (affectedRows == 0) {
    throw new SQLException("Creating inventory failed, no rows aff
  }

  generatedKeys = prepStmt.getGeneratedKeys();
  if (generatedKeys.next()) {
    mId = generatedKeys.getLong(1);
  } else {
    throw new SQLException(
        "Creating inventory failed, no generated key obtained.");
  }
} finally {
  if (generatedKeys != null) {
    try {
      generatedKeys.close();
    } catch (final SQLException logOrIgnore) {
    }
  }
  if (prepStmt != null) {
    try {
      prepStmt.close();
    } catch (final SQLException logOrIgnore) {
    }
  }
}
```

} ``
```

Batching SQL Statements

Basic Transactions

- autoCommit is normally on - Basic Transactions
    - single execution per transaction When it's off
    - `con.commit();`
    - `con.rollback();`
- Transactions Tutorial

Final Project

When you have completed all phases you will have a Java Application which performs these tasks (in order):

- Loads *stock_level.csv* into a inventory system, in a persistent database of your choice
- The order is read from a JSON file (*order.json*)
- An `Order` object from the read data
- The `OrderManager` `process`es the order as follows:
    - Tests that the items are in stock in a database by calling the `InventoryManager`'s `stockLevel(productId)`
    - If the stock on hand is >= the required quantity for the line item then
        - Decrements the stock level by calling the `InventoryManager`'s `decrementStock(productId)`
    - Otherwise it marks the `LineItem` as out_of_stock (all or nothing for each line item)
- Once the order and all the line items have been processed:
- The `Order` is passed to the `ReceiptPrinter` and a nicely formatted reciept is printed (*reciept.txt*)
- Exports *stock_level_updated.csv* to the file system with the updated stock levels read from the database

Background Topics

CSV Parsing

- opencsv

```
<dependency>
  <groupId>net.sf.opencsv</groupId>
  <artifactId>opencsv</artifactId>
  <version>2.3</version>
</dependency>
```

JSON

/articles/java/json-1973242.html)

- Strings in Switch were only added in java 7 :(
- json.org

```
<dependency>
  <groupId>org.json</groupId>
  <artifactId>json</artifactId>
  <version>20140107</version>
  <scope>compile</scope>
</dependency>
```

- [The Object Model API vs The Streaming API](http://www.oracle.com/technetwork

Logging API

- java.util.logging
- `mLogger = Logger.getLogger(App.class.getName());`
- See boxes/jsonweather

DbUnit

- Help test your database code with DbUnit
- Getting Started
- Allows importing of data and schemas easily

```
<dependency>
  <groupId>org.dbunit</groupId>
  <artifactId>dbunit</artifactId>
  <version>2.5.0</version>
  <scope>test</scope>
</dependency>
```

H2 Database

- Simple in memory database very useful for testing!

- `static final String JDBC_DRIVER = org.h2.Driver.class.getName();`
- `static final String JDBC_URL = "jdbc:h2:mem:inventory;DB_CLOSE_DELAY=-1";`

```
  <dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>1.4.179</version>
    <scope>test</scope>
  </dependency>
```

Sample Data

Sample Stock Data

stock_level.csv

```
PRODUCT_ID,STOCK_LEVEL
12,3
14,2
13,2
```

Sample Input

order.json

```
{
"timestamp":"20140709194738",
"store":{"id":"42", "name":"San Francisco" },
"customer":{ "first_name":"Bob", "last_name":"Williams", "rewards_
"items":[
  { "id":"12", "name":"Flat Screen TV", "price":500, "quantity":1
  { "id":"14", "name":"Playstation 4", "price":399, "quantity":1 }
  { "id":"13", "name":"Playstation Controller", "price":29, "quant
]
}
```

Sample Output

reciept.txt

```
  Bob Williams (Y18123AS)
  San Francisco (42)
  July 9th, 2014
  ---------------
  $500.00 - SOLD: 1 @ Flat Screen TV
  $399.00 - SOLD: 1 @ Playstation 4
  OUT_OF_STOCK: 4 @ Playstation Controller
  --------------
  Order Total: $899.00
  --------------
  Thank you for your business
```

Expectations

- Create a populate a local database with data from a CSV file (*stock_level.csv*)
- Write Unit Tests for the components
- Package the Application into Executable Jar File format (Research Required)
- Takes two command line arguments
    - orderfile - The file to read
    - recieptfile - The file to write
- Reads in *orderfile* (*order.json*) JSON File (representing a customer order)
    - Find and use an available Parser
- Creates Order Related Objects Based on the JSON File Contents
    - Order ? and what else might you need?
- Writes out *receiptfile* a nicely formatted receipt describing the order
- Has JUnit tests
    - Unit tests: for functional Units
    - Try you hand at an integration test
- Bonus: Using Logging

Learning Objectives Review

- Get Set Up for JavaSE 6 Development
- Know how to compile, run and debug Java applications
- Know how to create an application from scratch
- Understand the following java topics:
    - Classes and Objects
    - Primitive Data Types
    - Arrays
    - Collections
    - Loop Structures
    - Returning Values
    - Passing Arguments
    - Branching Structures
    - Exception Handling
    - Basic File Access
    - Stream based IO
- Understand the following Object Oriented Concepts:
    - Abstraction
    - Encapsulation
    - Constructors
    - Static versus Instance
    - Inheritance
    - Polymorphism
- Understand how to use Eclipse IDE for Java
    - Code Short-cuts
    - Formatting Rules
- Understand How To use JDBC to:
    - Connect to a Database
    - Query a Database
- Understand How To use JUnit 4:
    - Test existing code
    - Test First with TDD
    - Code Coverage

Bonus Topics

(Bonus)Connect to Oracle

- Maven?
- System Dependency
- Get Drivers
- Loading the Driver

```
try {
   Class.forName("oracle.jdbc.driver.OracleDriver");
}
catch(ClassNotFoundException ex) {
   System.out.println("Oracle JDBC Driver Unavailable");
   System.exit(1);
}
```

- Getting the Connection

```
String URL = "jdbc:oracle:thin:@hostname:port Number:databaseName"
String USER = "scott";
String PASS = "tiger"
Connection connection = DriverManager.getConnection(URL, USER, PAS
```

(Bonus)Enumerations

- Enumerations (StringTokenizer)

(Bonus)Annotations

- Writing Annotations
- Reading Annotations

(Bonus)Reading XML (SAX vs DOM)

- XML Processing

(Bonus)NumberFormat

- Jenkov NumberFormat

(Bonus)Regular Expressions

- Patterns
- String#matches

(Bonus)BigDecimal

(Bonus)Properties

- Demo boxes/checkout

(Bonus)Locales i18n

- Demo boxes/externalstrings
- ResourceBundles
- NumberFormat
- DecimalFormat

(Bonus)Java Thread Locks

- Java Processes
    - `jps`
- Java Thread Dump
    - `jstack`
- JConsole
    - `jconsole`
    - JConsole
    - Threads -> Detect Deadlock (com/skillbox/boxes/threads/LockerApp)

(Bonus) Recursion

- Rewrite the Fibonacci Exercise with recursion

(Bonus)Environment

- Properties
- System Properties

(Bonus)Serialization

- Object Streams
- `Serializable` Marker interface

(Bonus)Networking

- Java HTTP Networking
  - java.net.HttpURLConnection

(Bonus)Web Service

- Stand-alone Web Service
  - wsgen

(Bonus)Web Client

(Bonus)JavaDoc

- Simple example

(Bonus) Multiple Threads

- Where possible simplify your life by using your custom objects on single threads, otherwise:
- Concurrency
- `synchronized` - Ensure only a single thread is active at a given time
- `notify` and `wait`
- When threads are holding the object's monitor (Inside a `synchronized` block)...
- Example
- wait() and notify()

(Bonus)Garbage Collection Options

- Pete Freitag

(Bonus)Applets

- Applets

(Bonus) NIO/IO Java 7

- IO / NIO differences

Table of Contents