ORACLE
UNIVERSITY

Java Fundamentals

Activity Guide

X95176GC10

Edition 1.0 | October 2016

Learn more from Oracle University at **oracle.com/education/**

ORACLE

Authors

Nick Ristuccia
Eric Jendrock
Michael Williams
Luz Elena Peralta Ayala
Eduardo Moranchel Rosales

Editors

Aishwarya Menon
Nikita Abraham
Raj Kumar
Vijayalakshmi Narasimhan
Aju Kumar

Graphic Designer

Maheshwari Krishnamurthy

Publishers

Giri Venugopal
Srividya Rameshkumar
Raghunath M

Table of Contents

Project Management

Encapsulation and Conditionals

Dates, Arrays, Loops, and Inheritance

Interfaces and Exceptions

RESTful Applications

Application Deployment

Practice 1: Introduction to Scrum and Agile Development

Overview

There are no practices associated with Lesson 1.

Tasks

Relax.

Practice 2-1: Configuring Your IDE and JDK

Overview

In this practice, you install and configure your IDE and Java development environments. The Netbeans+JDK bundle is available for free from the Oracle Java website.

Tasks

1. Download and install the combined NetBeans and Java Development Kit (JDK) bundle from oracle.com/technetwork/java/javase/downloads.
2. Open a terminal or Command Prompt window and type `java` to see the Java Help menu.
    - On Windows, the terminal window is called the command prompt window. You start by opening the Start menu and typing `command` or `cmd` in the "Search programs and files" field of the Start menu
3. In the terminal or command prompt window, type `java -version` to check the Java version of your JDK installation.
4. Set your JAVA_HOME environment variable. It may be required by some software.
    - **On Windows**:
        a. Open Control Panel > System > Advanced System Settings > Environment Variables.
        b. In the Environment Variables dialog box, select New under System variables.
        c. Type `JAVA_HOME` for the Variable name, type the path to the folder into which the JDK was installed (for example, `C:\Program Files\Java\jdk1.8.0_71`) for the Variable value.
        d. Click OK.
    - **On MacOS:** Enter the following command in a terminal window to set the JAVA_HOME environment variable:
      `export JAVA_HOME="$(/usr/libexec/java_home -v 1.8)"`
5. Add JAVA_HOME/bin to your Windows Path variable.
    - The NetBeans+JDK installation added a path to only three of the JDK executables. The Path variable should be modified to include all of the JDK executables. Under the System variables in the Environment Variables dialog box, locate the Path variable and click Edit.
    - The NetBeans+JDK installation added "C:\ProgramData\Oracle\Java\javapath;" to the beginning of the Path. Replace it with "%JAVA_HOME%\bin;" without the quotes and click OK.
    - Click OK to close the Environment Variables dialog box, and then click OK to close the System Properties dialog box.

Practice 2-2: Installing Git

Overview

In this practice, you install Git and run a test to see if it installed properly.

Tasks

1. Download Git from https://git-scm.com/downloads and install it.
    - The webpage also shows you where to find GUI clients to help you with the most common operations. Download and install a GUI client (optional) if you want to use one after you master the command line.
2. On Windows, select the first option, **Use Git from Git Bash only**, during the installation.
3. To run Git on Windows, use the provided **git bash** or **git shell** application to open a command line with Git integration. Linux and MacOS both integrate Git with the default terminal. Start the default terminal.
4. Check your Git version:
   ```
   $ git --version
   ```
5. Display the Git help screen:
   ```
   $ git --help
   ```

Practice 2-3: Creating a Git Repository

Overview

In this practice, create a Git repository to store your files.

Tasks

1. Open a Git Bash or terminal window.
2. In your home directory, create a cloud directory: `mkdir cloud`
3. Change directory in the directory: `cd cloud`
4. Create a Git repository type: `git init`
5. The `cloud` directory is now a Git repository. To confirm this, type: `ls –a`
6. You should see that a `.git` directory has been created. Your repository is created.

Practice 3-1: Creating a New Project and Java Class

Overview

In this practice, you use NetBeans to create a new project and Java class.

Tasks

1. Create a new project called **practice03-1**.
    - Deselect the check box to create the main method. You'll write the main method yourself in the next practice.
2. Create a new **Java Class** file in this project.
    - Class name = `ShoppingCart`
    - Package name = `practice`

Practice 3-1 (Solution)

ShoppingCart.java:

```
public class ShoppingCart {


}
```

Practice 3-2: Creating a `main` Method

Overview

In this practice, you manually enter a main method that prints a message to the console.

Tasks

1. Continue editing **practice03-1** or open **practice03-2**.
2. In the code editor, add the main method structure to the ShoppingCart class.
3. In the code block of the main method, use a `System.out.println` method to print "Welcome to the Shopping Cart!".
4. Save your program.
5. Click the **Run** button to test program. 
   - Select **practice.ShoppingCart** as the main class.

Practice 3-2 (Solution)

ShoppingCart.java:

```
public class ShoppingCart {

    public static void main(String[] args) {
        System.out.println("Welcome to the Shopping Cart!");
    }
}
```

Practice 3-3: Checking a Project into Git

Overview

In this practice, you will set up Git and add a NetBeans project to your repository.

Configure Git

Before you commit changes to Git, you must configure your name and email address to identify your commits in the repository. Enter the following commands in a Git Bash or Terminal window:

1. To set your name: `git config –-global user.name "Your Name"`
2. To set your email address: `git config –-global user.email your-email@address`
3. To confirm that the values have been set: `git config –-global -l`
4. The output should display the values you just set.

Note: This sets your name and email address for all Git projects on this system. However, if you need to override these settings for a specific project, repeat the above steps in the project while omitting the `--global` option.

Copy and Build NetBeans Project

With Git configured, prepare to check in the project from Practice 3-2 into your repository.

1. Open Git Bash or a Terminal window.
2. Change into the `cloud` directory that you created in the previous lesson.
3. Create a directory for this lesson: `mkdir 03`
4. Change into the directory that you just created: `cd 03`
5. Copy the project from Practice 3-2 into this directory.
6. Open the project with NetBeans.
7. Right-click the project name and select **Rename**.
8. Rename the project to **Practice_03-03**. Select the **Also Rename Project Folder** option.
9. Click the **Rename** button. This should rename the project, the project directory, and all configuration files.
10. Build the project.
11. Run the project.
12. Verify that the project ran correctly. Proceed to the next section.

Set up the `.gitignore` File

With a version control system like Git, only the project source files should be versioned. Project class files and JAR files should not be included. Let's examine what will be committed to the Git repository.

1. Change back into the `cloud` directory: `cd ..`
2. Enter the following command: `git add -n .`
3. You should see results like the following:

```
add '.gitignore'
add '03/Practice_03-03/build.xml'
add '03/Practice_03-03/manifest.mf'
add '03/Practice_03-03/nbproject/build-impl.xml'
add '03/Practice_03-03/nbproject/genfiles.properties'
```

```
add '03/Practice_03-03/nbproject/project.properties'
add '03/Practice_03-03/nbproject/project.xml'
add '03/Practice_03-03/src/practice/ShoppingCart.java'
```

Note: Adding `-n` to the `git add` command performs a dry run. It lists the files that will be added to the repository if the option is not specified. It is a good practice to use this option before adding any files to make sure your `.gitignore` file is set up correctly.

4. Notice that we performed a build and a run. This should have created a `build` and `dist` directory. Check those directories to see if they contain any files.
5. The `build` directory contains class files and more. The `dist` directory contains a JAR file. But why are they not included in the add?
6. If you have moved out of the cloud directory, change back to the cloud directory: `cd ..`
7. Type the following command in the window: `cat .gitignore`
8. You should see results like the following:
```
/03/Practice_03-2_Solution/nbproject/private/
/03/Practice_03-03/nbproject/private/
/03/Practice_03-03/dist/
/03/Practice_03-03/build/
```
9. When NetBeans opens a project in a Git repository, it automatically writes rules to ignore the `build` and `dist` directories. Notice that rules have been written to ignore the `nbproject/private` directory as well. Thus, when you do an add, these directories are ignored.
10. Everything looks ready to commit.

Commit Your Changes to Git

With your Git and repository all configured, you are ready to commit your files into Git.

1. Change into the `cloud` directory if you are not there already.
2. Type: `git status`
3. Git shows that there are files and directories that have not been added yet.
4. Add the source files and `.gitignore` file to the repository: `git add`
5. There is no output from the add command.
6. To check that the files are added, type: `git status`
7. Now you should see all of the source files listed and flagged with "new file". However, no files have been saved to the repository for version tracking at this point.
8. To commit the files to the repository and begin version tracking, type: `git commit -m "Initial Commit"`
9. Git responds with something like this.
```
[master (root-commit) d2d1d6c] Initial commit
 8 files changed, 1606 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 03/Practice_03-03/build.xml
 create mode 100644 03/Practice_03-03/manifest.mf
 create mode 100644 03/Practice_03-03/nbproject/build-impl.xml
 create mode 100644 03/Practice_03-03/nbproject/genfiles.properties
 create mode 100644 03/Practice_03-03/nbproject/project.properties
 create mode 100644 03/Practice_03-03/nbproject/project.xml
 create mode 100644 03/Practice_03-03/src/practice/ShoppingCart.java
```

10. Your files are now checked in for version tracking.
11. Check the status of the repository: `git status`
12. You should get a response similar to this:
```
On branch master
nothing to commit, working directory clean
```

13. You can now detect changes to you checked in files with: `git status`
14. To commit new version use: `git commit -a -m "Your Notes here"`
15. The `-a` option automatically includes files that have changed since the last commit.

Practice 3-3 (Solution)

There are not specific solution files for this exercise.

Practice 3-4: Creating a Java Project and Checking It into Git

Overview

Use your new skills to create a NetBeans Java project that prints out "Hello World". Check the project into your Git repository.

Tasks

Perform the following tasks to create your project and check it into Git.

1. Create a new NetBeans project named HelloApp in the `cloud/03` directory.
2. Make the project a Java application.
3. Create a main class named `homework.Main`.
4. In the `Main` class, add a Java statement to print out the message "Hello World!".
5. Add and commit the new source files to your Git repository.
6. Observe the output from the following:
    a. The NetBeans output from running your program
    b. `git status` before adding the new project
    c. `git status` after adding the new project
    d. `git commit`
    e. `git status` after the commit

Practice 3-4 (Solution)

Your output should look similar to the following:

```
## Run
run:
Hellow World!
BUILD SUCCESSFUL (total time: 2 seconds)

## Git Status before Commit

git status

On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   .gitignore

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        03/HelloApp/

no changes added to commit (use "git add" and/or "git commit -a")


## git status after an add

git status

On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   .gitignore
        new file:   03/HelloApp/build.xml
        new file:   03/HelloApp/manifest.mf
        new file:   03/HelloApp/nbproject/build-impl.xml
        new file:   03/HelloApp/nbproject/genfiles.properties
        new file:   03/HelloApp/nbproject/project.properties
        new file:   03/HelloApp/nbproject/project.xml
        new file:   03/HelloApp/src/homework/Homework3_Solution.txt
        new file:   03/HelloApp/src/homework/Main.java


## git commit -a -m "Add hello project"

git commit -a -m "Add hello project"

[master 7b1f3b6] Add Hello project
 9 files changed, 1622 insertions(+), 1 deletion(-)
 create mode 100644 03/HelloApp/build.xml
```

```
 create mode 100644 03/HelloApp/manifest.mf
 create mode 100644 03/HelloApp/nbproject/build-impl.xml
 create mode 100644 03/HelloApp/nbproject/genfiles.properties
 create mode 100644 03/HelloApp/nbproject/project.properties
 create mode 100644 03/HelloApp/nbproject/project.xml
 create mode 100644 03/HelloApp/src/homework/Homework3_Solution.txt
 create mode 100644 03/HelloApp/src/homework/Main.java

##  git status after everything

git status

On branch master
nothing to commit, working directory clean
```

Practice 4-1: Using `String` Variables

Overview

In this practice, you declare, initialize, and concatenate `String` variables and literals.

Tasks

1. Open the project **Practice_04-1**.
2. Declare and initialize two String variables: `custName` and `itemDesc`.
3. Declare a String variable called `message`. Do not initialize it.
4. Assign the `message` variable with a concatenation of the `custName` and `itemDesc`. Include a String literal that results in a complete sentence (for example, "Mary Smith wants to purchase a Shirt").
5. Print `message` to the System output.

Practice 4-1 (Solution)

ShoppingCart.java:

```
public class ShoppingCart {
    public static void main(String[] args){
        // Declare and initialize String variables.
        // Do not initialize message yet.
        String custName = "Mary Smith";
        String itemDesc = "Shirt";
        String message;

        // Assign the message variable
        message = custName +" wants to purchase a " +itemDesc;

        // Print and run the code
        System.out.println(message);
    }
}
```

Practice 4-2: Using and Manipulating Numbers

Overview

In this practice, you declare and initialize numeric variables, and concatenate Strings with numbers.

Tasks

1. Continue editing **Practice_04-1** or open **Practice_04-2**.
2. Declare and initialize numeric fields: `price` (double) `tax` (double), and `quantity` (int). Also declare a double called `total`, but do not initialize it.
3. Change the `message` variable to include `quantity` (for example, "Mary Smith wants to purchase 1 Shirt.").
4. Calculate total by multiplying price * quantity * tax.
5. Print a message showing the total cost (example: "Total cost with tax is: 25.78.").

Practice 4-2 (Solution)

ShoppingCart.java:

```
public class ShoppingCart {
    public static void main(String[] args){
        // Declare and initialize String variables.
        // Do not initialize message yet.
        String custName = "Mary Smith";
        String itemDesc = "Shirt";
        String message;

        // Declare and initialize numeric fields: price, tax, and quantity.
        // Declare a total field but do not initialize it.
        double price = 29.99;
        int quantity = 2;
        double tax = 1.04;
        double total;

        // Assign the message variable
        message = custName +" wants to purchase " +quantity +" " +itemDesc;
        System.out.println(message);

        // Print and run the code
        total = quantity * price * tax;
        System.out.println("Total cost with tax: "+total);
    }
}
```

Practice 5-1: Using `if` Statements

Overview

In this practice, you declare, initialize, and concatenate `String` variables and literals.

Tasks

1. Open the project **Practice_05-1**.
2. Use an `if` statement to test the quantity of the item:
   - `if` it is > 1, concatenate an `'s'` to `message` so that it indicates multiple items.
3. Declare a boolean, `outOfStock`.
4. Use an `if | else` statement to test if the item is out of stock:
   - `if` item is out of stock, inform the user that the item is unavailable.
   - `else`, print the message and total cost.
5. Run the program with `outOfStock = true`.
6. Run it again with `outOfStock = false`.

Practice 5-1 (Solution)

ShoppingCart.java:

```
public class ShoppingCart {
    public static void main(String[] args){
        String custName = "Mary Smith";
        String itemDesc = "Shirt";
        String message;

        double price = 29.99;
        int quantity = 2;
        double tax = 1.04;
        double total;

        message = custName+" wants to purchase "+quantity +" " +itemDesc;
        total = quantity * price * tax;

        // Test quantity and modify message if quantity > 1.
        if (quantity > 1){
            message = message +"s";
        }

        // Declare outOfStock variable and initialize it.
        boolean outOfStock = false;

        // Test outOfStock and notify user in either case.
        if (outOfStock){
            System.out.println(itemDesc +" is out of stock.");
        }
        else{
            System.out.println(message);
            System.out.println("Total cost with tax: " +total);
        }
    }
}
```

Practice 5-2: Using an Array

Overview

In this practice, you declare and initialize a `String` array to hold item descriptions. Then you experiment with accessing the array.

Tasks

1. Open the project **Practice_05-2**.
2. Declare a `String` array and initialize it with four elements.
    - Each element represents a different item description ("Shirt", for instance).
3. Change `message` to show how many items the customer wants to purchase.
    - **Hint:** Use the `.length` property of your array.
4. Print just one element in the array.
    - What happens if you use index number 4?

Practice 5-2 (Solution)

ShoppingCart.java:

```
public class ShoppingCart {
    public static void main(String[] args){
        String custName = "Mary Smith";
        String message;

        // Declare and initialize the items String array with
        // 4 item descriptions.
        String[] items = {"Shirt", "Socks", "Scarf", "Belt"};

        // Change message to show the number of items purchased.
        message = custName +" wants to purchase " +items.length +" items.";
        System.out.println(message);

        // Print an element from the items array.
        System.out.println(items[2]);
        //System.out.println(items[4]);

    }
}
```

Practice 5-3: Using a Loop to Process an Array

Overview

In this practice, you loop through the array of item descriptions, printing each element.

Tasks

1. Continue editing **Practice_05-2** or open **Practice_05-3**.
2. Create a `for` loop that iterates through the array of item descriptions, displaying each element.
3. Precede the list of elements with the message: "Items purchased: "

Practice 5-3 (Solution)

ShoppingCart.java:

```
public class ShoppingCart {
    public static void main(String[] args){
        String custName = "Mary Smith";
        String message;

        String[] items = new String[4];
        items[0] = "Shirt";
        items[1] = "Belt";
        items[2] = "Scarf";
        items[3] = "Skirt";

        message = custName +" wants to purchase " +items.length +" items.";
        System.out.println(message);

        // Iterate through and print out the items from the items array
        System.out.println("Items purchased: ");
        for (String item : items ){
           System.out.print(item + ", ");
        }
    }
}
```

Practice 6-1: Creating the Item Class

Overview

In this practice, you create the Item class and declare public fields.

Tasks

1. Open the project **Practice_06-1**.
2. Create the Item class as a plain **Java class**.
3. Declare public fields for `ID` (int), `descr` (String), `price` (double), and `quantity` (int). You will not be able to test the Item class until Practice 6-2.

Practice 6-1 (Solution)

Item.java:

```
public class Item {
        public int itemID;
        public String desc;
        public double price;
        public int quantity;
}
```

Practice 6-2: Modifying the `ShoppingCart` to Use `Item` fields

Overview

In this practice, you declare and instantiate two variables of type Item in the ShoppingCart class and experiment with accessing properties and calling methods on the object.

Tasks

1. Continue editing **Practice_06-1** or open **Practice_06-2**.
2. Create a new **Java Main Class** called ShoppingCart. This class contains a single main method. The rest of this practice is spent modifying ShoppingCart.java.
3. Declare and instantiate two objects of type Item. Initialize only the `desc` field in each, using different values for each.
4. Print the description for each item and run the code.
5. (Optional) Above the code that prints the descriptions, assign item2 to item1. Run it again.

Practice 6-2 (Solution)

Item.java:

```
public class Item {
        public int itemID;
        public String desc;
        public double price;
        public int quantity;
}
```

ShoppingCart.java:

```
public class ShoppingCart {

    public static void main(String[] args) {
        // Declare and initialize 2 Item objects
        Item item1, item2;
        item1 = new Item();
        item2 = new Item();

        // Print both item descriptions and run code
        item1.desc = "Shirt";
        item2.desc = "Pants";

        // Assign one item to another and run it again.
        item1 = item2;
        System.out.println("Item1: " + item1.desc);
        System.out.println("Item2: " + item2.desc);
     }
}
```

Practice 6-3: Describing Objects and Classes

Overview

Welcome to the workforce! We have an exciting first project for you. Our newest client is a regional soccer league. They're asking us to create an application for their league that will keep track of statistics and rankings within an all-play-all league. I want you on this project.

Sometime in the future, we'll need to evolve the program to eventually encompass client-server web services. But for now, just worry about writing a program that represents the basic objects in this application: players, teams, goals, games, and the league. Good luck!

Tasks

Create a new project. Write classes to model the essential objects in the soccer league application: players, teams, goals, and games. Carefully consider which Java data types best represent the properties of each object. Make sure that these fields are public. Instantiate objects and assign values to their properties from the `League` class.

`Player` Class

Each player has a name.

`Team` Class

Each team has a name. A team also contains a roster of its players.

`Goal` Class

The application must keep track of goals scored. A goal can be described by the team who scored it, the player who scored it, and the time it was scored. Represent time as a `double`.

`Game` Class

The application must keep track of games played. A game can be described by its home team, its away team, and each goal that may have been scored.

`League` Class

This class contains a single main method. The main method simulates the league by instantiating objects and assigning values.

Instantiate and name at least six players and two teams. To remember which players play for which team, you may find it helpful to have player names and their team names start with the same letter. For example ***G***eorge Eliot plays for The ***G***reens; ***R***obert Service plays for the ***R***eds. Assign each player to a team. Print the roster for each team. Your code should be able to print every team member's name regardless of the size of the team.

Have the two teams play each other in a high-scoring game (at least three goals total). For each goal scored, print the time it was scored, the player on the roster who scored it, and their team. Your code should be able to print every goal regardless of the number of goals scored in a game.

When you have finished, make sure your work is committed to the repository that you created earlier.

Your code should produce similar output:

```
Greens:
George Eliot
Graham Greene
Geoffrey Chaucer

Reds:
Robert Service
Robbie Burns
Rafael Sabatini

Goals:
Goal scored after 12.0 mins by Geoffrey Chaucer of the Greens
Goal scored after 23.0 mins by Graham Greene of the Greens
Goal scored after 55.0 mins by Robert Service of the Reds
```

Practice 7-1: Using the `indexOf()` and `substring()` Methods

Overview

In this practice, you use the `indexOf()` and `substring()` methods to get just the customer's first name and display it.

Tasks

1. Open the project **Practice_07-1**.
2. Use the `indexOf()` method to get the index for the space character (" ") within `custName`. Assign it to `spaceIdx`.
3. Use the `substring()` method and the `spaceIdx` to get the first name portion of `custName`. Assign it to `firstName`. Print `firstName`.

Practice 7-1 (Solution)

ShoppingCart.java:

```
public class ShoppingCart {
    public static void main (String[] args){
        String custName = "Steve Smith";
        String firstName;
        int spaceIdx;

     // Get the index of the space character (" ") in custName.
        spaceIdx = custName.indexOf(" ");

     // Use the substring method to parse out the first name and print it.
        firstName = custName.substring(0, spaceIdx);
        System.out.println(firstName);
    }
}
```

Practice 7-2: Instantiating a `StringBuilder` Object

Overview

In this practice, you instantiate a `StringBuilder` object, initializing it to `firstName` using the `StringBuilder` constructor.

Tasks

1. Open the project **Practice_07-2** or continue editing the previous practice.
2. Instantiate a `StringBuilder` object (`sb`), initializing it to `firstName`, using the `StringBuilder` constructor.
3. Use the `append` method of the `StringBuilder` object to append the last name back onto the first name. You can just use a String literal for the last name. Print the `StringBuilder` object and test your code. It should show the full name.
4. (Optional) Can you append the last name without using a String literal?

Practice 7-2 (Solution)

ShoppingCart.java:

```
public class ShoppingCart {
    public static void main (String[] args){
        String custName = "Steve Smith";
        String firstName;
        int spaceIdx;

        spaceIdx = custName.indexOf(" ");
        firstName = custName.substring(0, spaceIdx);
        System.out.println(firstName);

    //Instantiate and initialize sb to firstName.
        StringBuilder sb = new StringBuilder(firstName);

    // Put the full name back together, using StringBuilder append method.
    // You can just enter the String literal for the last name.
    // Print the full name.
        sb.append(" Smith");

    // (Optional) Can you append the last name without a String literal?
        sb.append(custName.substring(spaceIdx));
        System.out.println(sb);    }
}
```

Practice 7-3: Declaring a `long`, `float`, and `char`

Overview

In this practice, you experiment with the data types introduced in this lesson by declaring and initializing variables, and then casting one numeric type to another.

Tasks

1. Open the project **Practice_07-3**.
2. Declare a `long`, using the `L` to indicate a long value. Make it a very large number (in the billions).
3. Declare and initialize a `float` and a `char`.
4. Print the `long` variable with a suitable label.
5. Assign the `long` to the `int` variable. Correct the syntax error by casting the `long` as an `int`.
6. Print the `int` variable. Note the change in value when you run it.

Practice 7-3 (Solution)

ShoppingCart.java:

```
public class ShoppingCart {
    public static void main(String[] args) {
        int int1;

    // Declare and initialize variables of type long, float, and char.
        long long1 = 99_000_000_000L;
        float flt1 = 13.5F;
        char ch1 = 'U';

    // Print the long variable.
         System.out.println("long1: "+long1);

    // Assign the long to the int and print the int variable.
        int1 = (int) long1;
        System.out.println("Long assigned to int var: " + int1);
    }
}
```

Practice 7-4: Manipulating Text

Overview

Our client has made an interesting observation. Fans want to look up statistics on individual players, but they'll often misspell or can't entirely remember the last name of a player. They'll wonder, for instance, do the Reds have a player called Sabatini? Or, is the name Sabatine or Sabadini? It's Sab-something…

We'd like you to do a little experimenting in the main method. Please find a way to search a roster for a particular player when the search criterion is just part of his or her last name, like "sab". After you've found the player, print the proper spelling of his or her name.

Our client also tells us that half of their fans prefer to read a team's roster according to last name. Keep experimenting in the main method and find a way to display all players in a roster according to the format lastname, firstname.

There are several methods that you can use to manipulate Strings. Don't be afraid to consult the Java documentation if you need to find a suitable method or research more information to complete your task.

Tasks

Continue editing your code from Practice 6-3. Find a way to search a roster for a particular player when the search criterion is just part of their last name, like "sab". This substring could be at the beginning, middle, or end of the last name. Once you've found a last name that matches the search criterion, print the proper spelling of the player's full name.

Consult the Java documentation to find a suitable method to accomplish this task. This method should accept a String parameter and return a boolean. You might choose a method that uses regular expression String. Regular expressions are Strings that include special characters that have special meanings. This makes regular expressions a very powerful option. Assume that you are looking for a player with a last name that starts with the letters "Sab". This substring would occur after the first name but not stretch all the way to the end of the string. Therefore, the regular expression String will be this exact substring plus some additional special characters to "skip over" the first name and then to "skip over" the end of the string. A single dot matches any character, but you do not know how many you need. But if you follow a single dot with a *, this will give you the regular expression that you need—`.*Sab.*`

Display the names of all players on a team in the format lastname, firstname. Each player's name should appear on a separate line.

To accomplish this task, you may use Strings or StringBuilders. You may also choose to use the `indexOf()` and `substring()` methods covered in class, or research other methods from the Java documentation if you feel something else would be more suitable.

When you have finished, make sure your work is committed to the repository.

Your code should produce similar output:

```
Greens:
George Eliot
Graham Greene
Geoffrey Chaucer

Reds:
Robert Service
Robbie Burns
Rafael Sabatini

Goals:
Goal scored after 12.0 mins by Geoffrey Chaucer of the Greens
Goal scored after 23.0 mins by Graham Greene of the Greens
Goal scored after 55.0 mins by Robert Service of the Reds

Found Rafael Sabatini
Eliot, George
Greene, Graham
Chaucer, Geoffrey
```

Practice 8-1: Declaring a `setColor` Method

Overview

In this practice, you declare a `setColor()` method that takes a char as an argument, call the `setColor()` method on `item1`, and test this method with both a valid color and an invalid color.

Tasks

1. Import the project **Practice_08-1**.

In the Item class:

2. Declare a `setColor()` method that takes a char as an argument (a color code) and returns a boolean. Return false if the colorCode is ' '. Otherwise, assign the colorCode to the color field and return true.

In the ShoppingCart class:

3. Call the `setColor()` method on `item1`. If it returns true, print out `item1.color`. If it returns false, print an invalid color message.
4. Test the `setColor()` method with both a valid color and an invalid one.

Practice 8-1 (Solution)

Item.java

```
public class Item {
    char color;

    // Declare and code the setColor method.
    public boolean setColor(char colorCode){
        if (colorCode == ' '){
            return false;
        }
        else {
            this.color = colorCode;
            return true;
        }
    }
}
```

ShoppingCart.java:

```
public class ShoppingCart {
    public static void main (String[] args){
        Item item1 = new Item();

        // Call the setColor method on item1.
        // Print an appropriate message, depending upon the return value.
        if (item1.setColor('B')){
            System.out.println("Item1.color = " +item1.color);
        }
        else System.out.println("Invalid color");
    }
}
```

Practice 8-2: Overloading a `setItemFields()` Method

Overview

In this practice, you create an overloaded method in the Item class. Then you invoke the overloaded method from the `ShoppingCart` class.

Tasks

1. Import the project **Practice_08-2**.

In the Item class:

2. Write a `setItemFields()` method that takes three arguments and assigns them to the `desc`, `quantity`, and `price` fields. The method returns `void`.
3. Write an overloaded `setItemFields()` method that takes four arguments and returns an `int`. The method assigns all four fields. A `' '` (a single space) is an invalid value for a `colorCode` argument.
   - If the `colorCode` argument is invalid, return -1 without assigning the value.
   - If the `colorCode` is valid, assign the `colorCode` field and then assign the remaining fields by calling the three-argument method.

In the ShoppingCart class:

4. Call the three-argument `setItemFields()` method and then call `item1.displayItem()`.
5. Call the four-argument `setItemFields()` method. Check the return value.
   - If the return value < 0, print an invalid color code message
   - Otherwise, call `displayItem()`.

Practice 8-2 (Solution)

Item.java

```
public class Item {
    String desc;
    int quantity;
    double price;
    char colorCode = 'U';   //'U' = Undetermined

    public void displayItem() {
        System.out.println("Item: " +desc +", " +quantity +", "
                +price +", " +colorCode);
    }

    // Write a public 3-arg setItemFields method that returns void.
    public void setItemFields(String desc, int qty, double pr) {
        this.desc = desc;
        this.quantity = qty;
        this.price = pr;
    }

    // Write a public 4-arg setItemDisplay method that returns an int.
    public int setItemFields(String desc, int qty, double pr, char c) {
        if (c != ' ') {
            colorCode = c;
            this.setItemFields(desc, qty, pr);
            return 1;
        }
        else{
            return -1;
        }
    }
}
```

ShoppingCart.java:

```
public class ShoppingCart {
    public static void main (String[] args){
        Item item1 = new Item();

        // Call the 3-arg setItemFields method and then call displayItem.
        item1.setItemFields("Belt", 1, 29.50);
        item1.displayItem();

        // Call the 4-arg setItemFields method, checking the return value.
        int retcode = item1.setItemFields("Shirt", 1, 34.99, ' ');
        if (retcode < 0) {
            System.out.println("Invalid color code.  Item not added.");
        }
        else {
            item1.displayItem();
        }
    }
```

Practice 8-3: Using Methods

Overview

Did you have fun experimenting in the main method? Your efforts were very helpful, by the way. But now it's time to get serious. Writing lots of code in the main method is handy when you're experimenting, but the main method can quickly become complicated and messy. It's best to make the main method as small as possible. I want you to focus on getting much of your logic out of the main method. Instead, place this logic in methods throughout the various other classes you've written as a way of describing those classes' behaviors. This will also help your coworkers understand your code more easily.

Have you ever had to work with another engineer before? I want people to understand what it's like to read someone else's code. Communication and organization are important because things don't always go according to plan. Do you know Kenny from down the hall? He's written a utility class to help with this assignment. Your project should now contain a new package that contains his file `GameUtils.java`. If it doesn't, you'll have to create the `utility` package yourself and add his file yourself.

Tasks

Continue editing your code from Practice 7-4. You've already written code in the main method that creates two teams. Put this code into a method within the League class. It should accept no argument and return an array of the teams you've created. This method should not print anything.

You've already written code in the main method that creates a game. Put this code into a method within the League class. It should accept an array of teams as an argument and return an array of games that you've created. The array you'll return may contain only a single game for now (You can enhance this in a later practice). Assign the game's home and away teams based on the Team array that you passed into this method.

You've already written code in the main method for creating goals. However, creating goals isn't quite a behavior of a league. Rather, it's a behavior that occurs in a game. Write a `playGame()`method in the Game class. This method should accept an int argument for the maximum number of goals scored in a game. The method's purpose is to set the Goal[] field and shouldn't return any value. Create an array containing a random number of goals (no longer than the maximum number of goals argument). You can use `Math.random()` to help generate an appropriate random number. Then, call Kenny's GameUtils `addGameGoals()` method to set the statistics for each goal. Remember to set the proper import.

Most of the time, games will have a maximum score of 6. Overload the `playGame()` method to create a version that accepts no arguments and sets the maximum number of goals to 6. Make sure this method doesn't duplicate code.

You've also already written code in the main method that prints statistics for each goal scored. Again, this is a behavior that would more appropriately occur in a game rather than a league. Write a method that either prints a description of each goal, or returns a String that can be printed elsewhere. You may find the `.toString()` method useful if you choose to work with a StringBuilder.

From the main method, instantiate a league object. To create the teams and games, call the methods that you created. Play a game and print a description of that game. Remove any other code from the main method. You don't need to create methods for the logic that you wrote in Practice 7-4.

When you're finished, make sure your work is committed to the repository.

Your code should produce similar output:

```
Goal scored after 10.0 mins by Robert Service of The Reds
Goal scored after 55.0 mins by Geoffrey Chaucer of The Greens
Goal scored after 72.0 mins by George Eliot of The Greens
Goal scored after 77.0 mins by Rafael Sabatini of The Reds
```

Practice 9-1: Installing and Configuring Maven

Overview

In this practice, you install the Maven Java build tool.

Tasks

1. Download Maven (tar.gz or zip) from http://maven.apache.org/download.cgi.
2. Unzip the distribution archive to the directory in which you want to install Maven.
3. Add the `M2_HOME` environment variable, pointing to that directory.
4. Append the value `$M2_HOME/bin` to the `PATH` environment variable.
5. Open a command prompt and run `mvn --version` to verify that Maven is correctly installed.

Practice 9-1 (Solution)

There are no solution files for this practice.

Practice 9-2: Creating a New Maven Project

Overview

In this practice, you create a new Maven project using an archetype.

Tasks

1. Change to the `cloud` directory where your Git repository is stored.
2. Create a directory for this lesson: `mkdir 09`
3. Create an empty Maven project using the `maven-archetype-quickstart` archetype. Enter the following command:
   ```
   mvn archetype:generate -DgroupId=com.example -DartifactId=09-02-Hello -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
   ```
4. The command creates an empty Maven project. Examine the directory structure and note that an executable class is located at `com.example.App`. Now the `pom.xml` file must be configured for plug-ins.
5. Open your newly created project in NetBeans.
6. Edit the `pom.xml` file.
7. Add the following `properties` settings to the file before the `dependencies` section. This sets the Java version and encoding for the project.
   ```
   <properties>
       <java.version>1.8</java.version>
       <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
   </properties>
   ```
8. After the dependencies element, add elements for build and plug-ins.
   ```
   <build>
      <plugins>
          <!-- Your plugins go here -->
      </plugins>
   </build>
   ```
9. Add the configuration for the compiler plug-in to the plug-ins section.
   ```
   <plugin>
       <groupId>org.apache.maven.plugins</groupId>
       <artifactId>maven-compiler-plugin</artifactId>
       <version>2.3.2</version>
       <configuration>
           <source>1.8</source>
           <target>1.8</target>
       </configuration>
   </plugin>
   ```
10. Right-click the `pom.xml` file and select Format to fix the indentation for the file. Save the file.
11. Add the exec plug-in to the `pom.xml` file.
    ```
    <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>exec-maven-plugin</artifactId>
        <version>1.4.0</version>
        <executions>
            <execution>
    ```

```
                <goals>
                    <goal>exec</goal>
                </goals>
            </execution>
        </executions>
        <configuration>
            <mainClass>com.example.App</mainClass>
        </configuration>
    </plugin>
```

12. Add the JAR plug-in to the `pom.xml` file.

```
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <version>2.6</version>
      <configuration>
         <archive>
           <manifest>
             <mainClass>com.example.App</mainClass>
           </manifest>
         </archive>
      </configuration>
    </plugin>
```

13. Fix the formatting in NetBeans.
14. Save the `pom.xml` file.
15. You can test your project. To compile the project, type: `mvn clean compile`
16. To execute the application, type: `mvn exec:java`
17. To package the application, type: `mvn package`
18. To execute the packaged JAR file, type: `java -jar target/09-02-Hello-1.0-SNAPSHOT.jar`
19. Commit your project to the Git repository.

Practice 9-2 (Solution)

See the solutions directory for this practice to see a sample project.

Practice 9-3: Deploying Your Maven Project to Developer Cloud Service

Overview

In this practice, you connect your Git repository to Oracle Developer Cloud Service. Then you set up a build of your application on Oracle Developer Cloud Service.

Get your Git URL

To perform this practice, you need to get information from your Cloud account. Perform the following steps.

1. Log in to your cloud account.
2. Open the Service Console for Developer Cloud Services.
3. Select your project.
4. You should see a Git repository has been created for you. Copy the SSH URL from the repository to your clipboard. You will need the URL in the next section.

Connect Git to the Cloud

Git is a distributed version control system. You can share your code with other repositories by adding a Git remote.

1. To add a Git remote, first change into the `cloud` directory that is the root directory of your repository.
2. An empty Git repository has been created for you on Oracle Developer Cloud Service. You will add your cloud repository as a remote so you can push your changes to the cloud. To create a Git remote, use that URL in the following command to connect your Git repository to the cloud repository:
   ```
   git remote add origin https://yoururl
   ```
3. The command makes a remote connection named "origin" to the other repository. However, no data has been synced. To sync your data to the remote, use the `push` command:
   ```
   git push origin master
   ```
4. Now all the source code along with all the changes you have made to your repository are synced up to the cloud repository.

Note: In a typical Git scenario, instead of creating your repository, you would clone an existing repository. Then to sync your updates back to that repository you would use the `push` command.

Note: NetBeans will automatically add the new Maven target directory to the `.gitignore` file.

Build Your Maven Project in the Cloud

With your Git repository connected, you can set up a build for this project on Oracle Developer Cloud Service.

1. Log in to your Oracle Developer Cloud Service Account.
2. Navigate to your project.
3. Click the **Build** tab, and then click **New Job**.
4. In the New Job dialog box, enter a unique **Job Name** (for example, `lesson9`).
5. Select **Create a free-style job** to create a blank job that must be configured to run a build.
6. Click **Save**.
7. You are now in the **Configure build job** dialog box. Click **Save**.
8. You are returned to the **Job Detail Page**. Use the **Configure** button to configure each aspect of your build job.
9. Click **Configure**.
10. Click the **Main** tab.
    a. Edit the job name if it needs adjusting.
    b. Enter a description.
    c. Set the JDK to **JDK 8**.
11. Click the **Source Control** tab.

    a. Select **Git** as your repository.
    b. For **URL**, select the URL to your Git repository.
12. Click the **Build Steps** tab.
    a. Click **Add Build Step**.
    b. Select **Invoke Maven 3**.
    c. Set the goals to: `clean package`.
    d. Set the POM file location to: `09/09-02-Hello/pom.xml`
    e. Click **Add Build Step**.
    f. Select **Execute Shell**.
    g. For **Command**, enter: `java -jar 09/09-02-Hello/target/09-02-Hello-1.0-SNAPSHOT.jar`
13. Click **Post Build**.
    a. Select **Archive the artifacts**.
    b. Set **Files To Archive** to: **/*.jar
    c. Set **Compression Type** to **NONE**.
14. Click **Save**. You are now ready to build your application.
15. Click **Build Now**. Your application will be queued and built.
16. If there are any errors, look at the console output and correct them until the build is successful.
17. Examine the data generated from the build.
18. Download the generated archive once you achieve a successful build.

Practice 9-3 (Solution)

There are no solution files for this practice.

Practice 9-4: Creating a Maven Project and a Cloud Build

Overview

Use your new skills to create convert Practice 8-3 into a Maven project. Check your new project into Git and then commit your changes. Finally, create a build job on Oracle Developer Cloud Service.

Tasks

Perform the following tasks to complete this practice.

1. Convert your NetBeans project for Practice 8-3 into a Maven project.

   Hint: Create an empty Maven project. The replace the Java source generated by Maven with the source files you created in Practice 8-3.
2. Make sure that you can build, package and run your project locally.
3. Commit your project to your Git repository.
4. Connect your local Git repository to an empty Git repository assigned to you on Oracle Developer Cloud Service.
5. Push your local repository to the cloud.
6. Create a build job for your project in Oracle Developer Cloud Service.
7. Turn in:
   a. Your source code and Maven project.
   b. The name of the build job you created on Oracle Developer Cloud Service.

Practice 9-4 (Solution)

See the solutions directory for this practice to see a sample project.

Practice 10-1: Encapsulating a Class

Overview

In this practice, you encapsulate the `Customer` class.

Tasks

1. Open the project **Practice_10-1**.
2. Change access modifiers so that fields must be read or modified through public methods.
3. Allow the `name` field to be read and modified.
4. Allow the `ssn` field to be read but not modified (read only).

Practice 10-1 (Solution)

Customer.java

```
public class Customer {
    private String name;
    private String ssn;

    // Encapsulate this class.  Make ssn read-only.
    public String getName(){
        return name;
    }
    public String setName(String n){
        name = n;
    }

    public String getSSN(){
        return ssn;
    }
}
```

ShoppingCart.java:

```
public class ShoppingCart {
    public static void main (String[] args){
        // Declare, instantiate, and initialize a Customer object.

        // Print the customer object name.

    }
}
```

Practice 10-2: Creating a Constructor

Overview

In this practice, you add a constructor to the `Customer` class and create a new `Customer` object by calling the constructor.

Tasks

1. Continue editing **Practice_10-1** or open **Practice_10-2**.

In the Customer class:

2. Add a custom constructor that initializes the fields.

In the ShoppingCart class:

3. Declare, instantiate, and initialize a new Customer object by calling the custom constructor.
4. Test it by printing the Customer object name (call the `getName` method).

Practice 10-2 (Solution)

Customer.java

```
public class Customer {
    private String name;
    private String ssn;

    //Add a custom constructor
    public Customer(String name, String ssn){
        this.name = name;
        this.ssn = ssn;
    }

    public String getName(){
        return name;
    }
    public String setName(String n){
        name = n;
    }

    public String getSSN(){
        return ssn;
    }
}
```

ShoppingCart.java:

```
public class ShoppingCart {
    public static void main (String[] args){
        // Declare, instantiate, and initialize a Customer object.
        Customer cust1 = new Customer("Bob Miller", "555-44-3212");

        // Print the customer object name.
        System.out.println("Customer name: " +cust1.getName();
    }
}
```

Practice 10-3: Using Encapsulation

Overview

Taking your logic out of the main method and putting it into other methods is a great way to make your code shorter and more sophisticated. So is encapsulation. I'm sure you already know that bad things can happen when classes are allowed to freely manipulate each other's fields, and I'd like you to continue making your code more sophisticated by encapsulating the classes in the soccer league application.

It would be good too if you could add constructors that would set the fields of the `Game`, `Player`, and `Team` classes. That should make your code even more efficient!

Also, Kenny says you won't need his `getHomeTeam` method anymore.

Test your code by playing four games instead of one. A great program like this should be able to handle any number of games.

Tasks

Continue editing your code from Practice 9. When you have finished, make sure your work is pushed to the repository as a Maven project. Your code must:

- Encapsulate the fields of the `Game`, `Goal`, `Player`, and `Team` classes. Provide corresponding getter and setter methods for each field.
- Provide whatever constructors you feel are necessary to set the fields of the `Game`, `Player`, and `Team` classes
- Fix any errors that may occur elsewhere as a result of encapsulation
- Remove the `getHomeTeam` method found in the `GameUtils` class. Fix any errors that occur as a result.
- Print the results of four unique games instead of one. The code should accommodate printing any number of games.

Your code should produce similar output:

```
Goal scored after 29.0 mins by Graham Greene of The Greens
Goal scored after 36.0 mins by Robbie Burns of The Reds
Goal scored after 48.0 mins by Robbie Burns of The Reds
Goal scored after 77.0 mins by Geoffrey Chaucer of The Greens

Goal scored after 29.0 mins by Bertrand Russell of the Blues
Goal scored after 79.0 mins by Robert Service of the Reds

Goal scored after 19.0 mins by George Eliot of The Greens
Goal scored after 31.0 mins by Robert Service of The Reds
Goal scored after 56.0 mins by Rafael Sabatini of The Reds

Goal scored after 59.0 mins by Geoffrey Chaucer of The Greens
```

Practice 11-1: Using the Ternary Operator

Overview

In this practice, you use a ternary operator to duplicate the same logic shown in this `if/else` statement:

```
01      int x = 4, y = 9;
02      if ((y / x) < 3){
03          x += y;
04      }
05      else x *= y;
```

Tasks

1. Open the project **Practice_11-1** or create your own project with a **Java Main Class** named `TestClass`.
2. Use a ternary operator to perform the same logic as shown in the `if | else` construct.
3. Print the result of the ternary operator. Keep in mind, that the `if` statement changes the value of `x`, so the output of the second print statement will be different than the first.

Practice 11-1 (Solution)

TestClass.java:

```
public class TestClass {
    public static void main (String[] args){
        int x = 4, y = 9;
        if (y / x < 3) {
            x += y;
        }
        else {
            x *= y;
        }
        System.out.println("After if statement, x = " + x);

        // Use a ternary operator to perform the same logic as above.
        x = ((y / x) < 3) ? (x += y) : (x *= y);
        System.out.println("After ternary operator, x = " + x);
    }
}
```

Practice 11-2: Chaining `if` Statements

Overview

In this practice, you write a `calcDiscount` method that determines the discount for three different customer types.

Tasks

1. Open the project **Practice_11-2**.

In the Order class:

2. Complete the `calcDiscount` method so it determines the discount for three different customer types:
   - Nonprofits get a 10% discount if their order is > 900; otherwise they get a 5% discount.
   - Private customers get a 7% discount if their order is > 900; otherwise they get no discount.
   - Corporations get an 8% discount if their order is < 500; otherwise they get a 5% discount.

In the ShoppingCart class:

3. Use the main method to test the `calcDiscount` method

Practice 11-2 (Solution)

Order.java

```
public class Order {
    static final char CORP = 'C';
    static final char PRIVATE = 'P';
    static final char NONPROFIT = 'N';
    String name, stateCode;
    double total, discount;
    char custType;

    public Order(String name, double total, String state, char custType) {
        this.name = name;
        this.total = total;
        this.stateCode = state;
        this.custType = custType;
        calcDiscount();
    }

    public String getDiscount(){
        return Double.toString(discount) + "%";
    }

    // Code the calcDiscount method.
    public void calcDiscount() {
        if (custType == NONPROFIT){
            if (total > 900){
                discount = 10.00;
            }else discount = 5.00;
        }else if (custType == PRIVATE){
            if (total > 900){
                discount = 7.00;
            }else discount = 0;
        }else if (custType == CORP){
            if (total < 500){
                discount = 8.00;
            }else discount = 5.00;
        }
    }
}
```

ShoppingCart.java:

```
public class ShoppingCart {
    public static void main (String[] args){
        Order order = new Order("Rick Wilson",910.00,"VA",Order.NONPROFIT);
        System.out.println("Discount is: " +order.getDiscount());
    }
}
```

Practice 11-3: Using `switch` Constructs

Overview

In this practice, you change the `calcDiscount` method of the `Order` class to use a `switch` construct instead of the chained `if` construct. You may wish to just comment out the chained `if` statement so that you will be able to reference it in order to duplicate the logic.

Tasks

1. Continue editing **Practice_11-2** or open **Practice_11-3.**

In the Order class:

2. Rewrite `calcDiscount` to use a switch statement.
   - Use a ternary expression to replace the nested if logic.
   - For better performance, use a `break` statement in each case block.
   - Include a `default` block to handle invalid `custType` values.

In the ShoppingCart class:

3. Use the main method to test the `calcDiscount` method.

Practice 11-3 (Solution)

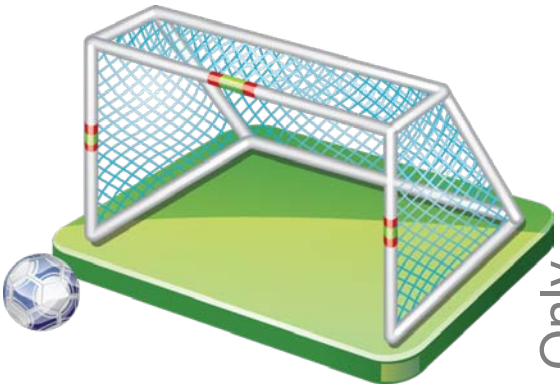
Order.java

```
public class Order {
    static final char CORP = 'C';
    static final char PRIVATE = 'P';
    static final char NONPROFIT = 'N';
    String name, stateCode;
    double total, discount;
    char custType;

    public Order(String name, double total, String state, char custType) {
        this.name = name;
        this.total = total;
        this.stateCode = state;
        this.custType = custType;
        calcDiscount();
    }

    public String getDiscount(){
        return Double.toString(discount) + "%";
    }

    public void calcDiscount() {
        // Code the calcDiscount method.
        switch(custType){
            case NONPROFIT:
                discount = (total > 900)? 10.00 : 5.00;
                break;
            case PRIVATE:
                discount = (total > 900)? 7.00 : 0;
                break;
            case (CORP):
                discount = (total < 500)? 8.00 : 5.00;
                break;
            default:
                System.out.println("Invalid custType");
        }
    }
}
```

ShoppingCart.java:

```
public class ShoppingCart {
    public static void main (String[] args){
        Order order = new Order("Rick Wilson",910.00,"VA",Order.NONPROFIT);
        System.out.println("Discount is: " +order.getDiscount());
    }
}
```

Practice 11-4: Using Conditionals



Overview

Our design department saw the latest build. They have a few suggestions. Because the soccer application is now playing more than one game at a time, they think a helpful way to differentiate one game from another would be to announce the matchup at the beginning of each game. Use the format “Away Team vs Home Team”. They also suggest printing the winner and final score at the conclusion of each game. This way, users won’t have to count and remember the significance of each goal scored. After all, there’s likely to be a ton of goals scored in the league.

We’re in a good position now to start keeping track of more league statistics. In the client’s soccer league, teams are awarded points based on victories. Users will need to see the total number of points and goals scored for each team. Find and congratulate the best team based on their points and goals.

Carefully consider how your classes will store, access, and analyze these statistics. You won’t need to edit the `gameUtils` class and you probably won’t need to edit any constructors. Remember to keep your fields private and your main method short. Because you’ll be keeping track of a bunch of numbers in if/else statements, you might find the debugging feature of your IDE helpful.

Tasks

Continue editing your code from Practice 10-3. When you’re finished, make sure your work is pushed to the repository as a Maven project. Your code must:

- Print the matchup at the beginning of each game description in the format “Away Team vs Home Team”.
- Print the winner and final score at the conclusion of each game description. Watch for matchups that end in a draw.
- Track the total number of points each team is awarded. Increment +2 points for a victory. Increment +1 point for a draw. Store this statistic in the `Team` class.
- Track the total number of goals each team has scored. Store this statistic in the `Team` class.
- Contain a method in the `League` class that:
    - Prints the total points and goals of each team. This doesn’t have to be done in any particular order. That’s done in Practice 14-3.
    - Finds and congratulates the winner of the league based on the following criteria:
        - The best team will have the highest number of points.
        - If point values are tied, use the number of goals scored as a tie-breaker.
        - If goals scored are also tied, allow for multiple winners.

Your code should produce similar output:

```
The Greens vs. The Reds
Goal scored after 18.0 mins by Robbie Burns of The Reds
Goal scored after 24.0 mins by Rafael Sabatini of The Reds
Goal scored after 75.0 mins by Robert Service of The Reds
The Reds win (0 - 3)

The Reds vs. The Greens
Goal scored after 0.0 mins by George Eliot of The Greens
Goal scored after 69.0 mins by George Eliot of The Greens
Goal scored after 75.0 mins by Geoffrey Chaucer of The Greens
Goal scored after 80.0 mins by Robert Service of The Reds
The Greens win (1 - 3)

The Greens vs. The Reds
Goal scored after 2.0 mins by Rafael Sabatini of The Reds
Goal scored after 18.0 mins by George Eliot of The Greens
Goal scored after 33.0 mins by Robert Service of The Reds
Goal scored after 54.0 mins by George Eliot of The Greens
Goal scored after 67.0 mins by Graham Greene of The Greens
Goal scored after 87.0 mins by Robert Service of The Reds
It's a draw! (3 - 3)

The Reds vs. The Greens
Goal scored after 14.0 mins by Graham Greene of The Greens
Goal scored after 17.0 mins by Rafael Sabatini of The Reds
Goal scored after 18.0 mins by Robert Service of The Reds
Goal scored after 63.0 mins by Graham Greene of The Greens
It's a draw! (2 - 2)



Team Points
The Greens : 4 : 8
The Reds : 4 : 9
This year’s champions are: The Reds!
```

Practice 12-1: Declaring a `LocalDateTime` Object

Overview

In this practice, you print and format today's date.

Tasks

1. Open the project **Practice_12-1** or create your own project with a **Java main class** named `TestClass`.
2. Declare a `LocalDateTime` object to hold the order date.
3. Initialize the object to the current date and time using the `now` static method of the class.
4. Print the `orderDate` object with a suitable label.
5. Format the `orderDate`, using the `ISO_LOCAL_DATE` static constant field of the `DateTimeFormatter` class.
6. Add the necessary package imports.
7. Print the formatted `orderDate` with a suitable label.

Practice 12-1 (Solution)

TestClass.java:

```
// import statements here:
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class TestClass {
    public static void main (String[] args){
        // Declare a LocalDateTime object
        x = ((y / x) < 3) ? (x += y) : (x *= y);
        System.out.println("After ternary operator, x = " + x);

        // Initialize the LocalDateTime object and print it.
        orderDate = LocalDateTime.now();
        System.out.println("Order date: " +orderDate);

        // Format the object using ISO_LOCAL_DATE; Print it.
        String fDate = orderDate.format(DateTimeFormatter.ISO_LOCAL_DATE);
        System.out.println("Formatted order date: " +fDate);
    }
}
```

Practice 12-2: Parsing the `args` Array

Overview

In this practice, you parse the `args` array in the main method to get the arguments and assign them to local variables.

Tasks

1. Open the project **Practice_12-2** or create your own project with a **Java main class** named `TestClass`.
2. Parse the `args` array to populate name and age.
   - If `args` contains fewer than two elements, print a message, telling the user that two arguments are required.
   - Remember that the age argument will have to be converted to an int.
     - **Hint:** Use a static method of Integer class to convert it.
3. Print the `name` and `age` values with a suitable label.

Practice 12-2 (Solution)

TestClass.java

```
public class TestClass {
    public static void main (String[] args){
        String name;
        int age;

        // Parse the args array to populate name and age.
        // Print an error message if fewer than 2 args are passed in.
        if (args.length < 2) {
            System.out.println("Invalid args. There must be 2 arguments");
        }
        else {
            name = args[0];
            age = Integer.parseInt(args[1]);
            System.out.println("Name = "+name+", Age = "+age);
        }
    }
}
```

Practice 12-3: Processing an Array of Items

Overview

In this practice, you code the `displayTotal` method of the `ShoppingCart` class so that it will iterate through the items array and print out the total for the Shopping Cart.

Tasks

1. Open the project **Practice_12-3.**

In the ShoppingCart class:

2. Code the `displayTotal` method. Use a standard for loop to iterate through the `items` array.
3. If the current item is out of stock (call the `isOutOfStock` method of the item), skip to the next loop iteration.
4. If it is not out of stock, add the item price to a total variable that you declare and initialize prior to the for loop.
5. Print the Shopping Cart total with a suitable label.

Practice 12-3 (Solution)

ShoppingCart.java

```
public class ShoppingCart {
    Items[] items = {new Item("Shirt",25.60),
                     new Item("WristBand",0),
                     new Item("Pants",35.99)};

    public static void main(String[] args) {
        ShoppingCart cart = new ShoppingCart();
        cart.displayTotal();
    }

    // Use a standard for loop to iterate through the items array,
    // adding up the total price.
    // Skip items that are back ordered. Display the Shopping Cart total.
    public void displayTotal(){
        double total = 0;
        for(int idx = 0; idx < items.length; idx++){
            if(items[idx].isBackOrdered())
          continue;
            total += items[idx].getPrice();
        }
        System.out.println("Shopping Cart total: " +total);
    }
}
```

Practice 12-4: Working with an `ArrayList`

Overview

In this practice, you create an `ArrayList` with at least three elements, add an element, and then remove an element.

Tasks

1. Open the project **Practice_12-4.**
2. Create a String ArrayList with at least three elements.
   - Be sure to add the correct import statement.
   - Print the ArrayList and test your code.
3. Add a new element to the middle of the list.
   - **Hint:** Use the overloaded `add` method that takes an index number as one of the arguments.
   - Print the list again to see the effect.
4. Test for a particular value in the `ArrayList` and remove it if it exists.
   - **Hint:** Use the `contains` method. It returns a boolean and takes a single argument as the search criteria.
   - Print the list again.

Practice 12-4 (Solution)

ShoppingCart.java

```
import java.util.ArrayList;

public class ShoppingCart {
    public static void main(String[] args) {
        // Declare, instantiate, and initialize an ArrayList of Strings.
        // Print and test your code.
        ArrayList<String> items = new ArrayList<>();
        items.add("Shirt");
        items.add("WristBand");
        items.add("Pants");

        // add (insert) another element at a specific index
        System.out.println(items);
        items.add(2, "Belt");
        System.out.println(items);

        // Check for the existence of a specific String element.
        // If it exists, remove it.
        if (items.contains ("Shirt")){
            items.remove("Shirt");
        }
        System.out.println(items);
    }
}
```

Practice 12-5: Iterating Through Data and Working with `LocalDateTime`

Overview

The client reminded us that new teams are formed at the beginning of every season. I need you to modify the soccer league application so that new teams are created from a master list of player names. All teams must have the same number of players. The same player cannot be used more than once in a season. Kenny has created `PlayerDatabase.java`, which should be placed in the `utility` package. This class currently contains a single String with possible names. Names are separated by commas. Kenny suggests consulting the Java documentation on `StringTokenizer` for help extracting each name from the String to create Player objects. You're welcome to choose between either an array or `ArrayList` to store data.

The code should be creating our game procedurally too. Create at least three teams that use Players generated from the `PlayerDatabase`. Ensure that all teams play each of their competitors. Each team should play once as a home team and once as an away team. Make sure that teams don't play themselves.

The design department would like to differentiate games even further by providing a date with each game description. Those dates might come in handy, because one of our client's requests is to make an announcement at the beginning of the league stating the duration of the season.

Tasks

Continue editing your code from Practice 11-4 and incorporate Kenny's work. When you have finished, make sure that your work is pushed to the repository as a Maven project.

- Include a `PlayerDatabase` class in the `utility` package.
    - You'll need to find a way to pull and store names from the String that Kenny provided. He recommends consulting the Java documentation on `StringTokenizer`. Your delimiter is a comma. This work would need to be done only once when a `PlayerDatabase` is first instantiated. You're welcome to ignore Kenny's recommendation if you have a different idea.
    - Include a `getTeam` method. It should accept an `int` number of players and return an array of Players of that length. Make sure player names are not repeated on the same team or across different teams.
- Rewrite the `createTeams` method of the `League` class to use the player database. Accept arguments for the names of each team and the number of players per team. Although this method must be able to handle a variable number of team names (assuming the team names are provided), create three teams with five players each for now. Don't worry about running out of names because Kenny provided plenty.
- Rewrite the `createGames` method of the `League` class to handle any number of teams. Iterate through the array of teams, ensuring that all teams play each other. Each team should play once as a home team and once as an away team. Make sure a team can't play against itself.
- Give each game a `LocalDateTime` property that is printed as part of the game description.
    - Modify the `createGames` method in the `League` class accordingly. For now, it's OK to make the first game take place now and then mark seven days between each subsequent game.
- Write a `getLeagueAnnouncement` method in the `League` class. This method must find the length of the season. That is, how many months and days exist between the first and last game of the season.
    - Kenny recommends consulting the Java documentation on the `Period` class. You'll find the `between`, `getMonths`, and `getDays` methods useful.
    - Convert `LocalDateTime` to `LocalDate` using the `toLocalDate` method.
    - Print this announcement before any game descriptions are printed.

Your code should produce similar output:

```
The league is scheduled to run for 1 month(s), and 4 day(s)

The Robins vs. The Crows
Date: 2017-03-01
Goal scored after 1.0 mins by Baroness Orczy of The Robins
Goal scored after 43.0 mins by Baroness Orczy of The Robins
Goal scored after 51.0 mins by Graham Green of The Crows
Goal scored after 75.0 mins by O. Henry of The Robins
The Robins win (3 - 1)

The Robins vs. The Swallows
Date: 2017-03-08
Goal scored after 10.0 mins by James Joyce of The Swallows
Goal scored after 88.0 mins by Frank O'Connor of The Swallows
The Swallows win (0 - 2)

The Crows vs. The Robins
Date: 2017-03-15
Goal scored after 40.0 mins by Graham Green of The Crows
Goal scored after 67.0 mins by Wilkie Collins of The Robins
Goal scored after 89.0 mins by William Shakespeare of The Crows
The Crows win (2 - 1)

The Crows vs. The Swallows
Date: 2017-03-22
Goal scored after 2.0 mins by Graham Green of The Crows
Goal scored after 11.0 mins by Dorothy Parker of The Crows
Goal scored after 59.0 mins by Graham Green of The Crows
Goal scored after 65.0 mins by William Shakespeare of The Crows
The Crows win (4 - 0)

The Swallows vs. The Robins
Date: 2017-03-29
Goal scored after 40.0 mins by Brian Moore of The Robins
Goal scored after 46.0 mins by Baroness Orczy of The Robins
Goal scored after 61.0 mins by James Joyce of The Swallows
Goal scored after 65.0 mins by O. Henry of The Robins
The Robins win (1 - 3)

The Swallows vs. The Crows
Date: 2017-04-05
Goal scored after 6.0 mins by Graham Green of The Crows
Goal scored after 15.0 mins by William Shakespeare of The Crows
Goal scored after 31.0 mins by Graham Green of The Crows
```

```
Goal scored after 43.0 mins by William Shakespeare of The Crows
The Crows win (0 - 4)


Team Points
The Robins : 2 : 7
The Crows : 3 : 11
The Swallows : 1 : 3
This year’s champions are: The Crows!
```

Practice 13-1: Creating a Subclass

Overview

In this practice, you create the `Shirt` class, which extends the `Item` class.

Tasks

1. Open the project **Practice_13-1**.
2. Examine the `Item` class. Pay close attention to the overloaded constructor and also the display method.
3. In the **practice_13_1** package, create a new class called `Shirt` that inherits from `Item`.
4. Declare two private `char` fields: `size` and `colorCode`.
5. Create a constructor method that takes three args (`price`, `size`, `colorCode`). The constructor should:
   - Call the two-arg constructor in the superclass
     - Pass a String literal for the `desc` arg ("Shirt").
     - Pass the price argument from this constructor.
   - Assign the `size` and `colorCode` fields.

In the ShoppingCart class:

6. Declare and instantiate a `Shirt` object, using the three-arg constructor.
7. Call the `display` method on the object reference. Where is the `display` method coded?

Practice 13-1 (Solution)

ShoppingCart.java:

```
public class ShoppingCart {
    public static void main (String[] args){
        // Instantiate a Shirt object.
        // Call display() on the object reference.
        Shirt shirt = new Shirt(25.99, 'M', 'P');
        shirt.display();
    }
}
```

Shirt.java:

```
public class Shirt extends Item {
    private char size;
    private char colorCode;

    public Shirt(double price, char size, char colorCode){
        super ("Shirt", price);
        this.size = size;
        this.colorCode = colorCode;
    }
}
```

Item.java:

```
public class Item {
    private int id;
    private String desc;
    private double price;
    static int nextId = 1;

    // Default constructor sets default values
    public Item(){
        setId();
        setDesc("No description assigned.");
        setPrice(0.00);
    }
    // Overloaded constructor takes description and price
    public Item(String desc, double price) {
        setId();
        setDesc(desc);
        setPrice(price);
    }
```

```
    public void display(){
        System.out.println("Item description: "+getDesc());
        System.out.println("\tID: "+getId());
        System.out.println("\tPrice: "+getPrice());
    }

    // Getter and Setter methods
    private void setId() {
        id = Item.nextId++;
    }
    public int getId() {
        return id;
    }

    public String getDesc() {
        return desc;
    }
    private void setDesc(String desc) {
        this.desc = desc;
    }

    public double getPrice() {
        return price;
    }
    private void setPrice(double price) {
        this.price = price;
    }
}
```

Practice 13-2: Overriding a Method in the Superclass

Overview

In this practice, you override the display method to show the additional fields from the Shirt class.

Tasks

1. Open **Practice_13-2** or continue editing **Practice_13-1**.

In the Shirt class:

2. Override the display method to do the following:
   - Call the superclass's display method.
   - Print the `size` field and the `colorCode` field.
3. Run the code. Do you see a different display than you did in the previous practice?

Practice 13-2 (Solution)

The ShoppingCart and Item classes are the same as they were in practice 13-1.

Shirt.java:

```
public class Shirt extends Item {
    private char size;
    private char colorCode;

    public Shirt(double price, char size, char colorCode){
        super ("Shirt", price);
        this.size = size;
        this.colorCode = colorCode;
    }

   // Override display() in the superclass to also show size and colorCode.
   // Avoid duplicating code.
    public void display(){
        super.display();
        System.out.println("\tSize: " +size);
        System.out.println("\tColor Code: " +colorCode);
    }
}
```

Practice 13-3: Using the `instanceof` Operator

Overview

In this practice, you use the `instanceof` operator to test the type of an object before casting it to that type.

Tasks

1. Open **Practice_13-3** or continue editing **Practice_13-2.**

In the Shirt class:

2. Add a public `getColor` method that converts the `colorCode` field into the corresponding color name.
   - Example: `'R' = "Red"`
   - Include at least three colorCode/color combinations.
3. Use a `switch` statement in the method and return the color String.

In the ShoppingCart class:

4. Modify the `Shirt` object's declaration so that it uses an `Item` reference type instead.
5. Call the `display` method of the object.
6. Use `instanceof` to confirm that the object is a `Shirt`.
   - If it is a `Shirt`:
     - Cast the object to a `Shirt` and call the `getColor` method, assigning the return value to a String variable.
     - Print out the color name using a suitable label.
   - If it is not a `Shirt`, print a message to that effect.
7. Test your code. You can test the non-Shirt object condition by instantiating an Item object instead of a `Shirt` object.

Practice 13-3 (Solution)

The Item class is the same as it was in practice 13-2.

ShoppingCart.java:

```
public class ShoppingCart {
    public static void main (String[] args){
        // Declare and instantiate a Shirt object using an
        // Item reference type instead
        Item item = new Shirt(25.99, 'M', 'P');

        // Test for non-Shirt object type
        //  Item item = new Item();

        // Call the display method on the item, then the getColor method
        item.display();
        if (item instanceof Shirt) {
            String color = ((Shirt) item).getColor();
            System.out.println("Color: " +color);
        }
        else{
            System.out.println("Item is not a Shirt.");
        }
    }
}
```

Shirt.java:

```
public class Shirt extends Item {
    private char size;
    private char colorCode;

    public Shirt(double price, char size, char colorCode){
        super ("Shirt", price);
        this.size = size;
        this.colorCode = colorCode;
    }

    public void display(){
        super.display();
        System.out.println("\tSize: " +size);
        System.out.println("\tColor Code: " +colorCode);
    }
```

```
    // Code a public getColor method that converts the colorCode to the
    // color name.
    // Use a switch statement.  Return the color name.
    public String getColor(){
        String color = "";
        switch (colorCode){
            case 'R':
                color = "Red";
                break;
            case 'G':
                color = "Green";
                break;
            case 'B':
                color = "Blue";
                break;
            case 'Y':
                color = "Yellow";
                break;
            default:
                color = "Invalid code";
        }
        return color;
   }
}
```

Practice 13-4: Creating a Game Event Hierarchy

Overview

Our client would eventually like to see many different types of game events, such as goals, possessions, kickoffs, passes, and fouls, printed with each game description. I'd like you to enhance the soccer league application so that it can accommodate many types of game events. Some of these game events may have additional properties that we'll need to track, but all game events must contain information regarding a Team, Player, and time (double). I'd like you to write code to support goals and possessions for now. Other types of game events can be created another day.

I'd also like you to completely rewrite the `playgame` method in the `Game` class. The goal of this method now is to generate `GameEvent` objects at random**.** This will replace the work done in `GameUtils.java`.

Print all game events in the `getDescription` method of the `Game` class. Modify the printout to differentiate each type of game event. It's OK if the printout is messy for now. You'll also notice that the code is interpreting possessions as goals. You can fix this next time, too.

Finally, rename any methods or variables in the `Game` class that now deal with `GameEvent` objects instead of `Goal` objects. This will address any confusion caused by misleading names.

Tasks

Continue editing your code from Practice 12-5. When you have finished, make sure your work is pushed to the repository as a Maven project.

- Create an abstract `GameEvent` class with encapsulated fields for Team, Player and time.
- Use inheritance to create the `Goal` class.
- Use inheritance to create the `Possession` class.
- Make sure that subclasses don't contain any unnecessary or duplicate fields and methods.
- Rewrite the `playgame` method of the `Game` class so that it iterates through all 90 minutes of a game. As you iterate, simulate a game by randomly creating game events by random players. Make sure that these events occur at a realistic pace. You may need to do some experimenting to find a reasonable frequency. This method does not need to take arguments. You may delete the overloaded version of this method, and delete Kenny's `GameUtils.java`.
- Alter the printout in the `gameDescription` method of the `Game` class to differentiate a `Goal` from a `Possession`.
    - You may choose to insert the current GameEvent object reference in the printout. This may lead to a messy printout, which includes the fully qualified class name, but this can be fixed later.
    - Your program might be counting possessions as goals. This can also be fixed later.
- Rename any methods or variables in the `Game` class that now deal with `GameEvent` objects instead of `Goal` objects. This may include:
    - Rename the `goals` field `gameEvents`. Change this field from a `Goal` to a `GameEvent` type.
    - Rename the `getGoals` method `getEvents`.
    - Rename the `setGoals` method `setEvents`.
    - Rename the `currGoal` variable `currEvent`.

Your code should produce similar output for the description of a single game:

```
The Robins vs. The Crows
Date: 2017-03-03
soccer.Possession@6ce253f1 after 5.0 mins by James Joyce of The Robins
soccer.Possession@53d8d10a after 10.0 mins by Emile Zola of The Robins
soccer.Possession@e9e54c2 after 40.0 mins by Baroness Orczy of The Crows
soccer.Possession@65ab7765 after 64.0 mins by Agatha Christie of The Crows
soccer.Goal@1b28cdfa after 76.0 mins by G. K. Chesterton of The Robins
soccer.Possession@eed1f14 after 88.0 mins by J. R. Tolkien of The Robins
The Robins win (4 - 2)
```

Practice 14-1: Converting an Array to an ArrayList

Overview

In this practice, you convert a `String` array to an `ArrayList` and manipulate list values.

Tasks

1. Open the project **Practice_14-1** or create your own project with a **Java main class** named `TestClass`.
2. Convert the `days` array to an `ArrayList`.
   - Use `Arrays.asList` to return a `List`.
   - Use that `List` to initialize a new `ArrayList`.
   - Preferably do this all on one line.
3. Iterate through the `ArrayList`, testing to see if an element is `"sunday"`.
   - If it is a `"sunday"` element, print it out, converting it to uppercase. Use `String` class methods:
     - `public boolean equals (Object o);`
     - `public void toUpperCase();`
   - Otherwise, print the day anyway, but not in uppercase.
4. After the `for` loop, print out the `ArrayList`.
   - While within the loop, was `"sunday"` printed in uppercase?
   - Was the `"sunday"` array element converted to uppercase?
   - Your instructor will explain what's going on in the next topic.

Practice 14-1 (Solution)

TestClass.java:

```
import java.util.ArrayList;
import java.util.Arrays;

public class TestClass {
    public static void main (String[] args){
        String[] days = {"monday","saturday","tuesday","sunday","friday"};

        // Convert the days array into an ArrayList
        // Loop through the ArrayList, printing out "sunday" elements in
        //   uppercase (use toUpperCase() method of String class)
        // Print all other days in lowercase
        // Print out the ArrayList

        ArrayList<String> dayList = new ArrayList(Arrays.asList(days));
        for (String s : dayList){
            if (s.equals("sunday")){
                System.out.println(s.toUpperCase());
            }
            else{
                System.out.println(s);
            }
        }
        System.out.println(dayList);

    }
}
```

Practice 14-2: Using the `Predicate` Lambda Expression

Overview

In this practice, you use the `removeIf()` method to remove all items of the shopping cart whose description matches some value.

Tasks

1. Open the project **Practice_14-2**.

In the ShoppingCart class:

2. Examine the code. As you can see, the items list has been initialized with two shirts and two pairs of trousers.
3. In the `removeItemFromCart` method, use the `removeIf` method (which takes a `Predicate` lambda type) to remove all items whose description matches the `desc` argument.
4. Print the items list. Hint: the `toString` method in the `Item` class has been overloaded to return the item description.
5. Call the `removeItemFromCart` method from the main method. Try different description values, including ones that will return `false`.
6. Test your code.

Practice 14-2 (Solution)

ShoppingCart.java:

```
public class ShoppingCart {
    ArrayList<Item> items = new ArrayList();

    public static void main (String[] args){
        ShoppingCart cart = new ShoppingCart();
        cart.fillCart();
        cart.removeItemFromCart("Trousers");
    }

    public void fillCart(){
        items.add(new Shirt(40.95,'M','R'));
        items.add(new Shirt(32.99,'M','Y'));
        items.add(new Trousers(59.99,34,'B',"Relaxed",'M'));
        items.add(new Trousers(75.50,8,'G',"Skinny",'F'));
    }

    public void removeItemFromCart(String desc){
        // remove all Trousers from the items list, then print out the list
        items.removeIf(i -> i.getDesc().equals(desc));
        System.out.println("Cart after removing Trousers: \n" +items);
    }
}
```

Trousers.java:

```
public class Trousers extends Item {
    private char size;
    private char gender;
    private String fit;

    public Trousers(double price, char size, char colorCode, String fit,
char gender){
        super ("Trousers", price, colorCode);
        this.setSize(size);
        this.setGender(gender);
        this.setFit(fit);
    }

    public void display(){
        super.display();
        System.out.println("\tSize: " +getSize());
        System.out.println("\tGender: " +getGender());
        System.out.println("\tFit: " +getFit());
```

```
    }

    public int getSize() {
        return size;
    }
    public void setSize(int size) {
        this.size = size;
    }

    public char getGender() {
        return gender;
    }
    public void setGender(char gender) {
        this.gender = gender;
    }

    public String getFit() {
        return fit;
    }
    public void setFit(String fit) {
        this.fit = fit;
    }
}
```

Shirt.java:

```
public class Shirt extends Item {
    private char size;

    public Shirt(double price, char size, char colorCode){
        super ("Shirt", price, colorCode);
        this.setSize(size);
    }

    public void display(){
        super.display();
        System.out.println("\tSize: " +getSize());
    }

    public int getSize() {
        return size;
    }
    public void setSize(int size) {
        this.size = size;
    }
}
```

Item.java:

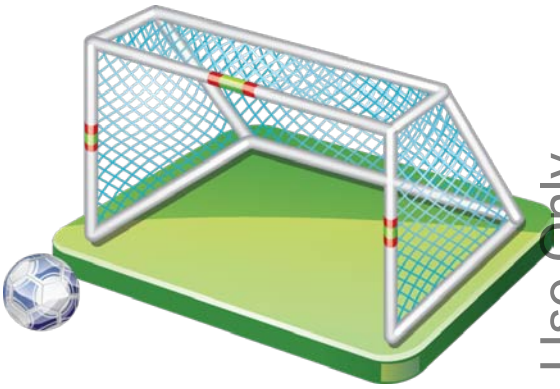
```
public class Item {
    private int id;
    private String desc;
    private double price;
    private char colorCode;
    static int nextId = 1;

    // Default constructor sets default values
    public Item(){
        setId();
        setDesc("No description assigned.");
        setPrice(0.00);
    }
    // Overloaded constructor takes description and price
    public Item(String desc, double price, char cCode) {
        setId();
        setDesc(desc);
        setPrice(price);
        setColorCode(cCode);
    }

    public void display(){
        System.out.println("Item description: "+getDesc());
        System.out.println("\tID: "+getId());
        System.out.println("\tPrice: "+getPrice());
        System.out.println("\tColor: "+getColorCode());
    }

    // Getter and Setter methods
    private void setId() {
        id = Item.nextId++;
    }
    public int getId() {
        return id;
    }

    public String getDesc() {
        return desc;
    }
    private void setDesc(String desc) {
        this.desc = desc;
    }
```

```
    public double getPrice() {
        return price;
    }
    private void setPrice(double price) {
        this.price = price;
    }

    public char getColorCode() {
        return colorCode;
    }
    private void setColorCode(char colorCode) {
        this.colorCode = colorCode;
    }

    public String toString(){
       return this.getDesc();
    }
}
```

Practice 14-3: Overriding and Interfaces

Overview

There were a few things that I said were OK to leave unfinished. I think now would be a good time to complete those features. If the game description printout still contains the fully qualified class name with a hex value, I'd like you to make the output look more user friendly. A `possession` should no longer be counted as a `goal`. And make sure teams are printed in the order of their final ranking at the end of the season. To help you sort through the teams, Kenny recommends letting the `Team` class implement the `Comparable` interface and consulting the Java documentation on this interface.



Tasks

Continue editing your code from Practice 13-4. When you're finished, make sure that your work is pushed to the repository as a Maven project.

- Make `GameEvent` announcements more user friendly in the game description printout.
  - Overload the `toString` method in the `Goal` class to return a more suitable description.
  - Overload the `toString` method in the `Possession` class to return a more suitable description.
- Modify the `getDescription` method in the `Game` class so that possessions aren't counted as goals. If necessary, adjust the frequency at which possessions or goals occur.
- Sort teams by rank by letting the `Team` class implement the `Comparable` interface.
  - Implement the `compareTo` method, which is used to compare the current `Team` with a `Team` passed as an argument. This method will accept a single `Object` argument and return an `int`. Remember, all Java objects automatically inherit from the `Object` class. This is where methods like `toString` and `equals` come from.
  - If the current team has fewer points, return 1.
  - If the current team has more points, return -1
  - If both teams have an equal number of points, examine goals instead.
- Modify the `showBestTeams` method in the `League` class to print the newly ordered teams.
  - Call `Arrays.sort(theTeams);` at the beginning of this method to sort the array of teams (this assumes all teams are stored in an array).
  - Remove any redundant code from this method.
  - Announce the league champions. In the case of a tie, announce multiple champions.

Your code should produce similar output:

```
<-- OUTPUT OMITTED -->

The Crows vs. The Robins
Date: 2017-03-23
Possession after 7.0 mins by W. B. Yeats of The Crows
Goal scored after 28.0 mins by Alan Patton of The Robins
Possession after 33.0 mins by George Eliot of The Crows
Goal scored after 47.0 mins by W. B. Yeats of The Crows
Goal scored after 63.0 mins by George Eliot of The Crows
Goal scored after 83.0 mins by W. B. Yeats of The Crows
The Crows win (3 - 1)

Team Points
The Crows : 4 : 8
The Swallows : 4 : 8
The Robins : 0 : 1
This year’s champions are: The Crows, The Swallows!
```

Practice 15-1: Catching an Exception

Overview

In this practice, you implement exception handling. Change a method signature to indicate that it throws an exception, and then catch the exception in the class that calls the method.

Tasks

1. Open the project **Practice_15-1**.

In the Calculator class:

2. Change the `divide` method signature so that it throws an `ArithmeticException`.

In the TestClass class:

3. Surround the code that calls the `divide` method with a `try/catch` block. Handle the exception object by printing it to the console.
4. Run the `TestClass` to view the outcome.

Practice 15-1 (Solution)

Calculator.java:

```
public class Calculator {
    public int add() {
        return x + y;
    }
    // This method could throw an ArithmeticException
    public double divide(int x, int y) throws ArithmeticException {
        return x / y;
    }
}
```

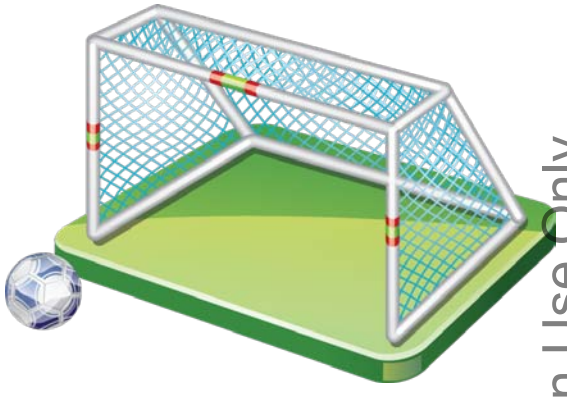
TestClass.java:

```
public class TestClass {
    public static void main (String[] args){
        Calculator calc = new Calculator();
        int addResult = calc.add(43, 79);
        System.out.println("Add Result: "+addResult);

        // Handle possible ArithmeticException
        try {
            double divResult = calc.divide(15, 0);
            System.out.println("Division Result: "+divResult);
        }
        catch (ArithmeticException ae){
            System.out.println(ae);
        }
    }
}
```

Practice 15-2: Adding Exception Handling

Overview

There is one last thing about the soccer league application that needs to be patched. I mentioned earlier that Kenny had supplied plenty of names, and that we shouldn't worry about running out of players to create teams. That's great because it allows the program to run without problems, but it might not reflect the reality of how the application will be used. Our client tells us that the number of people interested in the league and the size of teams may fluctuate each year. If the user tries to create teams when there aren't enough players, the application must catch this and report an error message describing the situation to the user. I'd like you to create a custom error message that describes the problem in a helpful way to both programmers and non-technical users.



Tasks

Continue editing your code from Practice 14-3. When you have finished, make sure that your work is pushed to the repository as a Maven project.

- Create a `PlayerDatebaseException` class in the `utility` package.
  - Make this a subclass of the `Exception` class.
  - Write a constructor that passes a helpful message to the superclass's constructor.
- Try creating teams such that more players are needed than are available.
  - Catch whatever exception may occur in the `getTeam` method of the `PlayerDatabase` class as a result.
  - When this exception is caught, throw a new `PlayerDatabaseException`.
- Throw the `PlayerDatabaseException` between methods as necessary, all the way up to the main method. Because the entire program is dependent on having fully-assembled teams, ensure that the rest of the program does not execute if the required number of teams cannot be fully assembled.
  - Hint: The `Exception` class contains the `printStackTrace` method, which may prove useful.

Your code should produce similar output when the exception occurs:

```
utility.PlayerDatabaseException: Not enough players in the database for the
teams requested.
    at utility.PlayerDatabase.getTeam(PlayerDatabase.java:39)
    at soccer.League.createTeams(League.java:57)
    at soccer.League.main(League.java:31)
```

Practice 16-1: Reading HTTP Headers

Overview

In this practice, use the cURL utility to read the HTTP headers of several common websites.

cURL

cURL (pronounced k-ur-l, originally stood for “see URL”) is a command-line tool for downloading files from a URL and supports a number of Internet protocols. It also makes an excellent tool for testing the REST operations of an application.

The following switches will be used for the practice:

-X Specifies the HTTP method to be used in the HTTP request.

-i Include the HTTP header in the output. This downloads the header along with any other data that is normally returned with the request.

-I Return the HTTP header only. In this case, an HTML page will not be included in the result.

-H Add an HTTP header to the request. You can specify as many of this options as you need.

-d Specify data to upload for methods like POST or PUT.

For more information:

- Git Bash – Type `curl –help`
- Unix – Type `man curl`

Examples

```
curl -X GET -i https://www.google.com
```

Displays the Google home page in the console

```
curl -X GET -I https://www.google.com
```

Display the HTTP headers returned from the Google home page.

Tasks

1. Open the Git Bash console.
2. Use the `curl` command to display the Google home page (https://www.google.com) and its headers.
3. Use the `curl` command to display just the HTTP headers returned from the Google home page (https://www.google.com).
4. Use the `curl` command to display just the HTTP headers for the following URLs:
   - `https://www.yahoo.com`
   - `http://www.amazon.com`
5. Compare the headers and note any differences.

Practice 16-1 (Solution)

The following is the sample output produced from executing the commands. The output will likely vary from one request to the next.

curl -X GET -i https://www.google.com

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--
0HTTP/1.1 200 OK
Date: Wed, 24 Feb 2016 16:57:59 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
P3P: CP="This is not a P3P policy! See
https://www.google.com/support/accounts/answer/151657?hl=en for more info."
Server: gws
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Set-Cookie: NID=76=noWVxnAE561REBbn-GztpsLwPEoser0gbXXWX__J-
lRteJ9Yzc5Ry2YnPejKm9g7QStr5UMS7fezBEZcvDtKN4UrpkUyGPklA93G54BxgnIqAM43S0DSsjR1
Iyu2e1l4D71bPKloQzE8LA; expires=Thu, 25-Aug-2016 16:57:59 GMT; path=/;
domain=.google.com; HttpOnly
Alternate-Protocol: 443:quic,p=1
Alt-Svc: quic="www.google.com:443"; ma=2592000;
v="30,29,28,27,26,25",quic=":443"; ma=2592000; v="30,29,28,27,26,25"
Accept-Ranges: none
Vary: Accept-Encoding
Transfer-Encoding: chunked

<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage"
lang="en"><head><meta content="Search the world's information, including
webpages, images
<!- More HTML text goes here, but not included to save space. -->
```

curl -X GET -I https://www.google.com

```
Executing: curl -X GET -I https://www.google.com
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--
0HTTP/1.1 200 OK
Date: Wed, 24 Feb 2016 17:02:19 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
P3P: CP="This is not a P3P policy! See
https://www.google.com/support/accounts/answer/151657?hl=en for more info."
Server: gws
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
```

```
Set-Cookie:
NID=76=qd1yjWIjVxY6oIG0FzrENvHvMvIj5pcay1MlIEJeRCMwWHWrYke7ted3Uxu8rEBJrrkDEosj
HfXTRdLoKm5Iy4GtBDtFeTn7WT3O7f-gKgGpzqYIUEz4iGjBov6jI0Nsnd_3bO15J2IGIA;
expires=Thu, 25-Aug-2016 17:02:19 GMT; path=/; domain=.google.com; HttpOnly
Alternate-Protocol: 443:quic,p=1
Alt-Svc: quic="www.google.com:443"; ma=2592000;
v="30,29,28,27,26,25",quic=":443"; ma=2592000; v="30,29,28,27,26,25"
Accept-Ranges: none
Vary: Accept-Encoding
Transfer-Encoding: chunked
```

curl -X GET -I https://www.yahoo.com

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--
0HTTP/1.1 200 OK
Date: Wed, 24 Feb 2016 17:03:30 GMT
P3P: policyref="http://info.yahoo.com/w3c/p3p.xml", CP="CAO DSP COR CUR ADM DEV
TAI PSA PSD IVAi IVDi CONi TELo OTPi OUR DELi SAMi OTRi UNRi PUBi IND PHY ONL
UNI PUR FIN COM NAV INT DEM CNT STA POL HEA PRE LOC GOV"
Strict-Transport-Security: max-age=2592000
X-Frame-Options: DENY
Vary: Accept-Encoding
Content-Type: text/html; charset=utf-8
Age: 0
Transfer-Encoding: chunked
Connection: keep-alive
Via: http/1.1 ir22.fp.ne1.yahoo.com (ApacheTrafficServer)
Server: ATS
Cache-Control: no-store, no-cache, private, max-age=0
Expires: -1
Y-Trace:
BAEAQAAAAAD6Jh3GgvZTFAAAAAAAAAAA7Ou6fjLmXG4AAAAAAAAAAAAFLIcLlvoUAAUshwuYNLKVyRr
UAAAAAA--
```

curl -X GET -I http://www.amazon.com

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--
0HTTP/1.1 200 OK
Date: Wed, 24 Feb 2016 17:09:15 GMT
Server: Server
Set-Cookie: skin=noskin; path=/; domain=.amazon.com
pragma: no-cache
x-amz-id-1: 0CG5584ZETZ4D0HH86T8
p3p: policyref="https://www.amazon.com/w3c/p3p.xml",CP="CAO DSP LAW CUR ADM
IVAo IVDo CONo OTPo OUR DELi PUBi OTRi BUS PHY ONL UNI PUR FIN COM NAV INT DEM
CNT STA HEA PRE LOC GOV OTC "
cache-control: no-cache
x-frame-options: SAMEORIGIN
```

```
expires: -1
Vary: Accept-Encoding,User-Agent
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
```

Practice 16-2: Working with Spring Boot and JSON Data

Overview

In this practice, you use `curl` to test a Spring Boot REST application. Instead of returning HTML, a Spring Boot REST application works with JSON data when communicating with an HTTP client.

Starting the Spring Boot Application

A sample application is provided for you to test REST requests. To start the application, perform the following steps.

1. Open a **Command Prompt** window.
2. Change directories into your `labs\solutions\16-02-sol` directory.
3. To compile the application, type the following command: `mvn clean compile`
4. To create the JAR file: `mvn package`
5. To run the application type: `java -jar target/16.2-customer-rs-16.2.0.jar`

The Spring Boot application should now launch and start listening on port 8080.

Getting a JSON Response

With the REST application running, REST calls can now be made to the application. Open a git Bash window to execute `curl` commands. For example, to look up customer number 100, the `curl` command and response would be:

```
curl -X GET -i http://localhost:8080/customers/100
```

And the response should similar to the following:

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Fri, 26 Feb 2016 23:24:27 GMT

{"id":100,"firstName":"George","lastName":"Washington","email":"gwash@example.c
om","city":"Mt Vernon","state":"VA","birthday":"1732-02-23"}
```

Notice the data is returned in a JSON format.

POST Data to Web Service (Add)

To add a customer to the REST service, you can make a POST call. The REST service both produces and consumes JSON data. To add a customer with an ID of 105, use the following command.

```
curl -X POST -i -H "Content-Type: application/json" -d '{"id":105,
"firstName":"Abigail","lastName":"Adams","email":"aadams@example.com","city":"B
raintree","state":"MA", "birthday":"1744-11-22"}'
http://localhost:8080/customers
```

Notice how the JSON data is enclosed in single quotes. If the request completes successfully, you should receive the following response:

```
HTTP/1.1 201 Created
Server: Apache-Coyote/1.1
Content-Length: 0
Date: Fri, 26 Feb 2016 23:31:05 GMT
```

Notice that the response code is in the 200 series, but not 200. In this case, it is 201 Created.

Tasks

With the basics covered, make the following REST calls to complete this practice.

1. Retrieve a list of all customers by making a GET call to: `http://localhost:8080/customers`
2. Using the PUT method, update customer 101 with the following JSON data.
   Hint: The syntax for update is almost exactly the same as add except the HTTP PUT method is used instead of POST.
   ```
   {"id":101, "firstName":"John
   Quincy","lastName":"Adams","email":"jqadams@example.com","city":"Braintree","s
   tate":"MA", "birthday":"1767-07-11"}
   ```
3. Delete customer 104.
   Hint: The syntax is almost exactly the same as a GET except the HTTP method DELETE is used instead.

Practice 16-2 (Solutions)

Get All Customers

```
curl -X GET -i http://localhost:8080/customers
```

Response

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Fri, 26 Feb 2016 23:40:16 GMT

[{"id":100,"firstName":"George","lastName":"Washington","email":"gwash@example.
com","city":"Mt Vernon","state":"VA","birthday":"1732-02-
23"},{"id":101,"firstName":"John","lastName":"Adams","email":"jadams@example.co
m","city":"Braintree","state":"MA","birthday":"1735-10-
30"},{"id":102,"firstName":"Thomas","lastName":"Jefferson","email":"tjeff@examp
le.com","city":"CharlottesVille","state":"VA","birthday":"1743-04-
13"},{"id":103,"firstName":"James","lastName":"Madison","email":"jmad@example.c
om","city":"Orange","state":"VA","birthday":"1751-03-
16"},{"id":104,"firstName":"James","lastName":"Monroe","email":"jmo@example.com
","city":"New York","state":"NY","birthday":"1758-04-
28"},{"id":105,"firstName":"Abigail","lastName":"Adams","email":"aadams@example
.com","city":"Braintree","state":"MA","birthday":"1744-11-22"}]
```

Update Customer 101

```
curl -X PUT -i -H "Content-Type: application/json" -d '{"id":101,
"firstName":"John
Quincy","lastName":"Adams","email":"jqadams@example.com","city":"Braintree","st
ate":"MA", "birthday":"1767-07-11"}' http://localhost:8080/customers/101
```

Response

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Length: 0
Date: Fri, 26 Feb 2016 23:48:09 GMT
```

Delete Customer 104

```
curl -X DELETE -i http://localhost:8080/customers/104
```

Response

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Length: 0
Date: Fri, 26 Feb 2016 23:53:07 GMT
```

Get all Data After Operations

If you retrieve all the customers after these operations your data should look something like this. Note: If you have to restart your Spring Boot application, any changes to data stored in the application are lost.

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Fri, 26 Feb 2016 23:55:44 GMT

[{"id":100,"firstName":"George","lastName":"Washington","email":"gwash@example.
com","city":"Mt Vernon","state":"VA","birthday":"1732-02-
23"},{"id":101,"firstName":"John
Quincy","lastName":"Adams","email":"jqadams@example.com","city":"Braintree","st
ate":"MA","birthday":"1767-07-
11"},{"id":102,"firstName":"Thomas","lastName":"Jefferson","email":"tjeff@examp
le.com","city":"CharlottesVille","state":"VA","birthday":"1743-04-
13"},{"id":103,"firstName":"James","lastName":"Madison","email":"jmad@example.c
om","city":"Orange","state":"VA","birthday":"1751-03-
16"},{"id":105,"firstName":"Abigail","lastName":"Adams","email":"aadams@example
.com","city":"Braintree","state":"MA","birthday":"1744-11-22"}]
```

Practice 16-3: Using Postman to Read HTTP Headers (Optional)

Overview

In this practice, use the Postman Chrome browser plug-in to read the HTTP headers of several common websites.

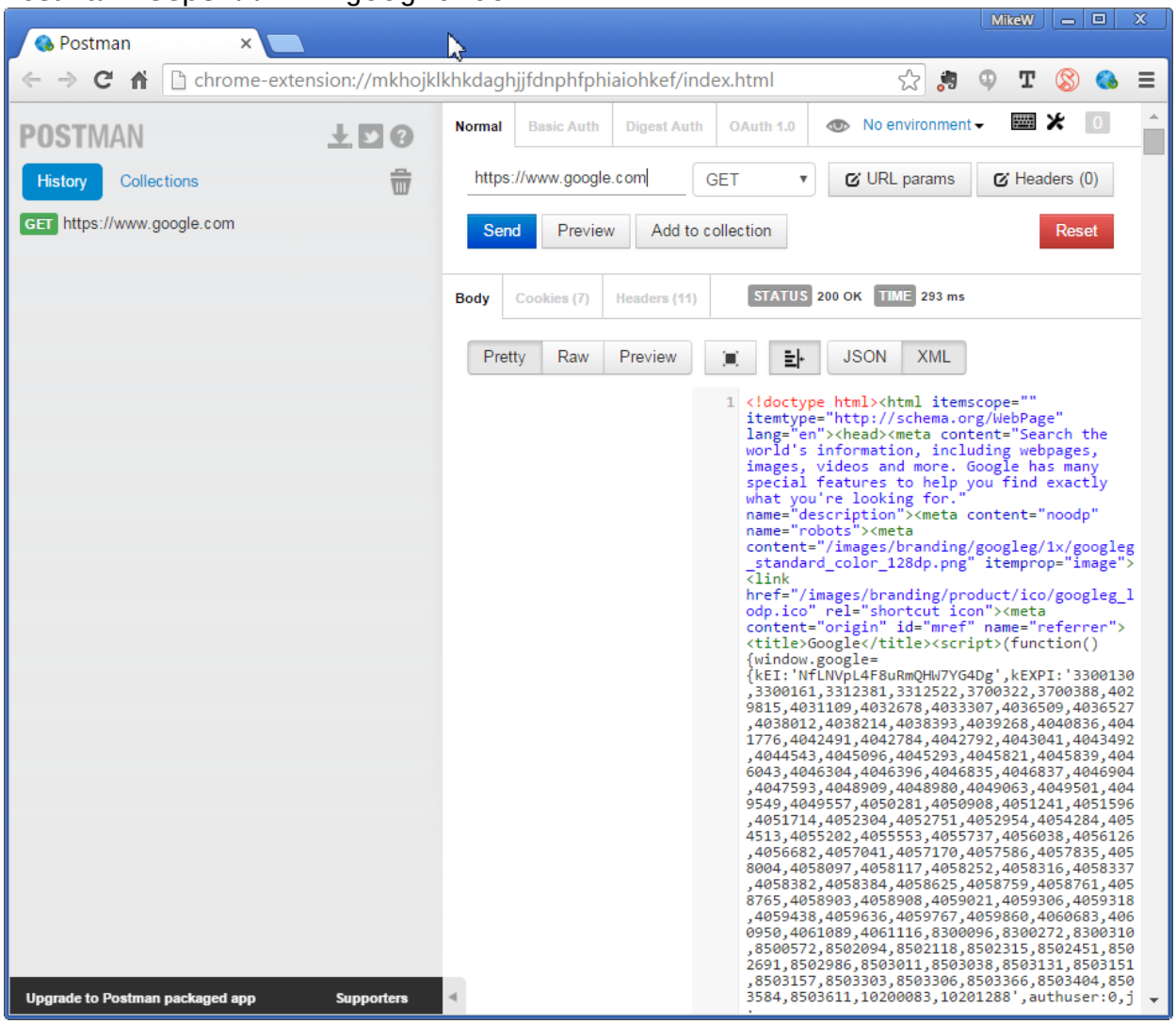
Tasks

1. Start the Chrome web browser. If you have not installed it, you can download and install it from: https://www.google.com/chrome/browser/desktop/
2. From the Chrome menu, select **Settings** and then **Extensions**. Scroll to the bottom of the page and click **Get More Extensions**. In the **Search** box, enter `Postman` and press **Enter**. Click the **Add to Chrome** button and install the plugin. This may require you to restart your browser.
3. Launch Postman from your toolbar.
4. Use Postman to display the Google home page (https://www.google.com) and its headers.
5. Use Postman to display just the HTTP headers returned from the Google home page (https://www.google.com).
6. Use the `curl` command to display just the HTTP headers for the following URLs:
   - `https://www.yahoo.com`
   - `http://www.amazon.com`
7. Compare the headers and note any differences.

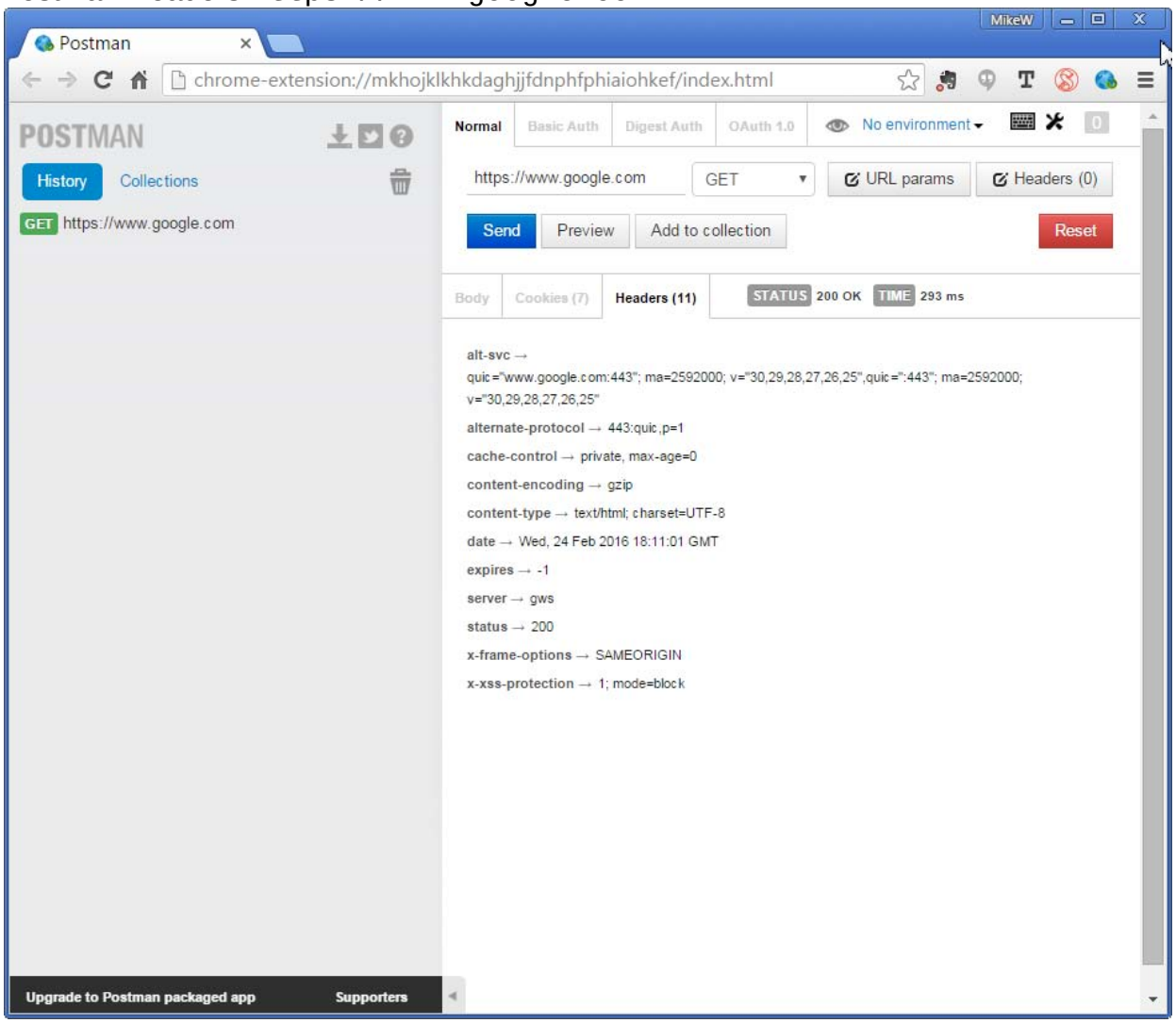
Practice 16-3 (Solution)

The following snapshots show the HTTP information returned in Postman. The HTTP output will likely vary from one request to the next.

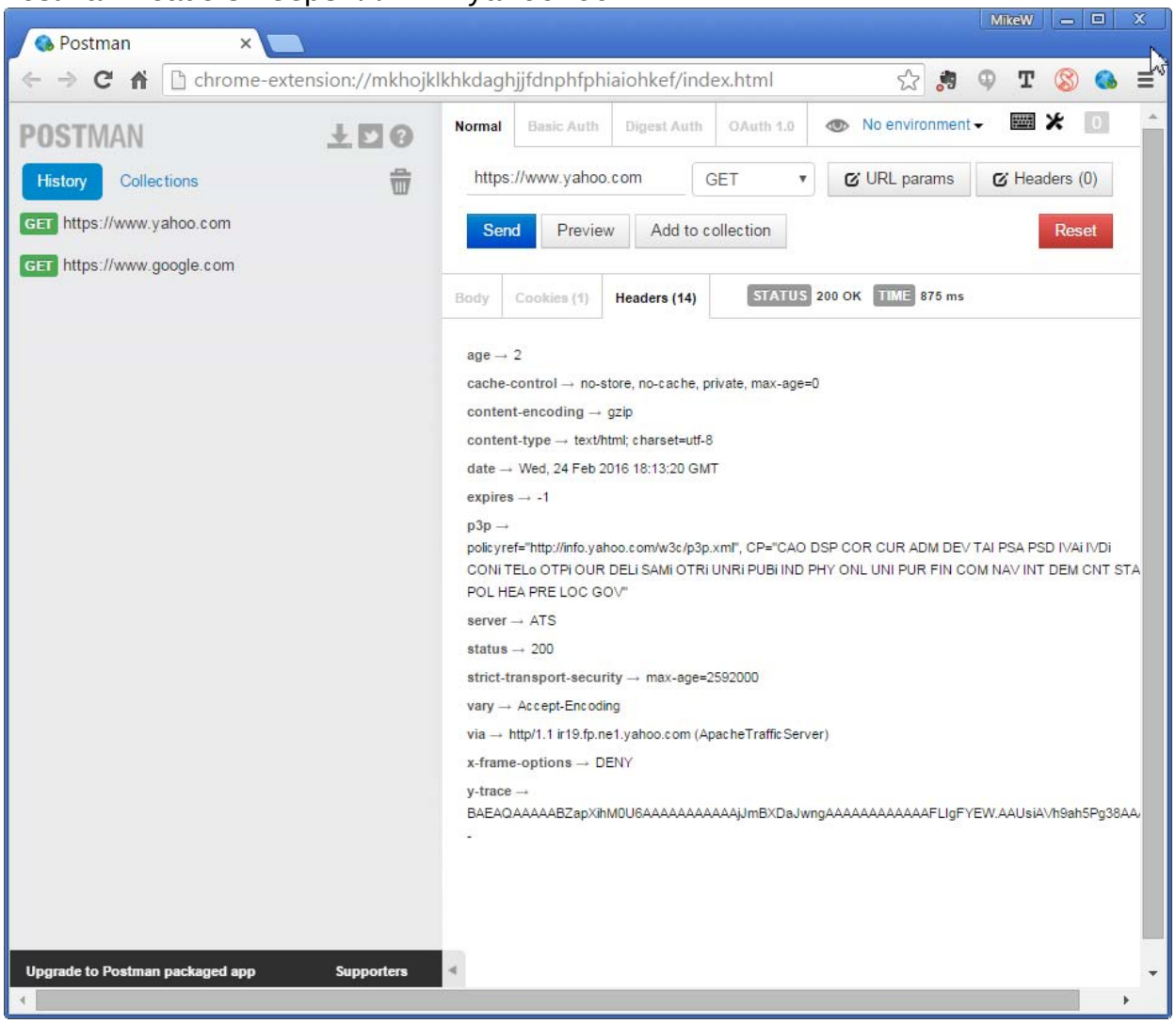
Postman `https://www.google.com`



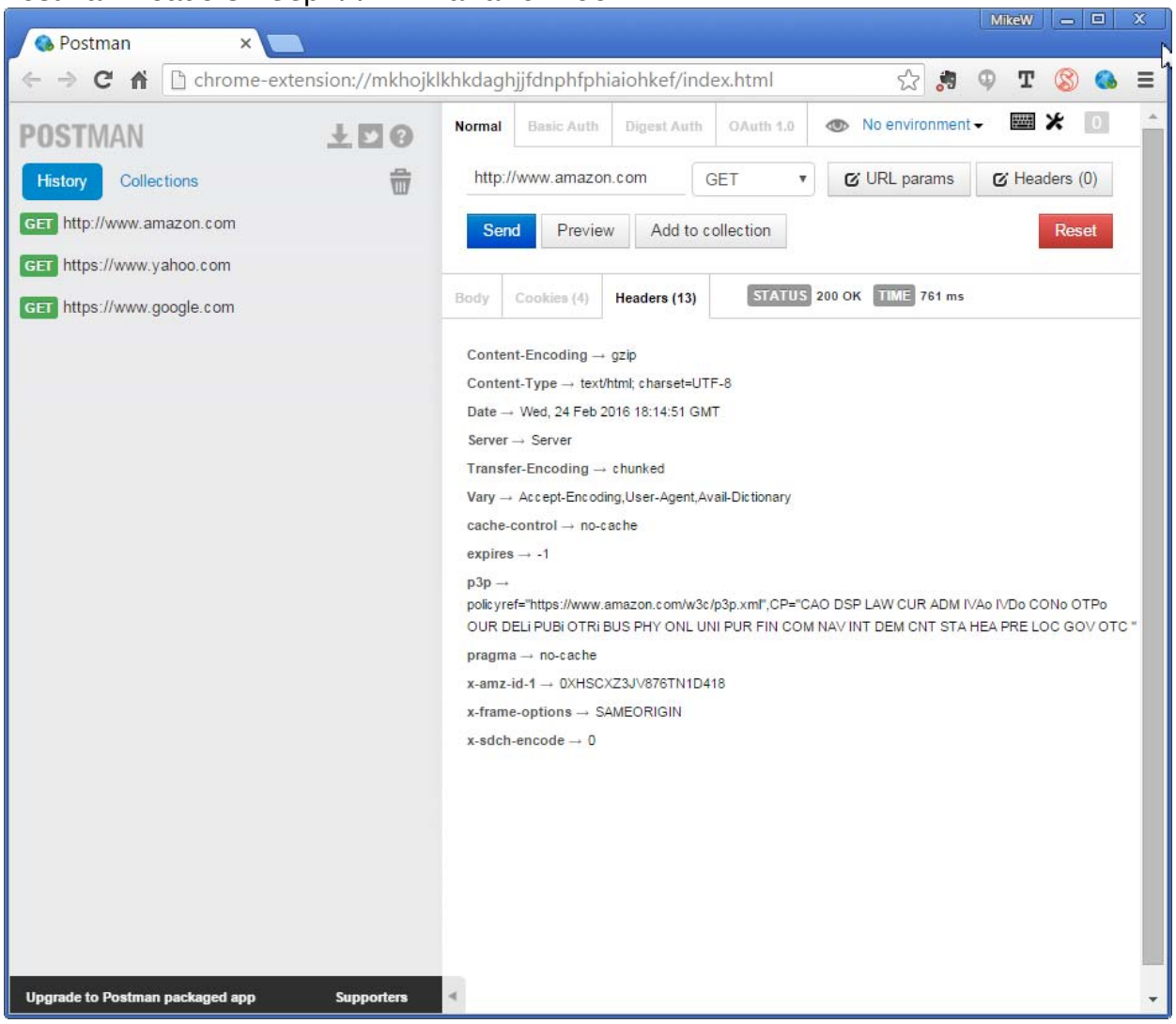
Postman Headers `https://www.google.com`



Postman Headers `https://www.yahoo.com`



Postman Headers `http://www.amazon.com`



Practice 16-4: Working with Spring Boot and JSON Data (Optional)

Overview

In this practice, you use Postman to test a Spring Boot REST application. Instead of returning HTML, a Spring Boot REST application works with JSON data when communicating with an HTTP client.

Starting the Spring Boot Application

A sample application is provided for you to test REST requests. To start the application, perform the following steps:

1. Open a **Command Prompt** window.
2. Change directories into your `labs\ex\16\16-02` directory.
3. To compile the application, type the following command: `mvn clean compile`
4. To create the JAR file: `mvn package`
5. To run the application type: `java -jar target/16.2-customer-rs-16.2.0.jar`

The Spring Boot application should now launch and start listening on port 8080.

Getting a JSON Response

With the REST application running, REST calls can now be made to the application. Open Postman in the Chrome browser. For example, to look up customer number 100, set the following values in Postman and then click **Send**:

```
Method: GET
URL: http://localhost:8080/customers/100
```

And the response should similar to the following:

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Fri, 26 Feb 2016 23:24:27 GMT

{"id":100,"firstName":"George","lastName":"Washington","email":"gwash@example.c
om","city":"Mt Vernon","state":"VA","birthday":"1732-02-23"}
```

Notice the data is returned in a JSON format.

POST Data to Web Service (Add)

To add a customer to the REST service, you can make a POST call. The REST service both produces and consumes JSON data. To add a customer with an ID of 105, use the following values and click **Send**:

```
Method: POST
Headers:
  Header: Content-Type
  Value: application/json
Raw: {"id":105,
"firstName":"Abigail","lastName":"Adams","email":"aadams@example.com","city":"B
raintree","state":"MA", "birthday":"1744-11-22"}
URL: http://localhost:8080/customers
```

Notice how the JSON data is enclosed in single quotes. If the request completes successfully, you should receive the following response:

```
HTTP/1.1 201 Created
Server: Apache-Coyote/1.1
Content-Length: 0
Date: Fri, 26 Feb 2016 23:31:05 GMT
```

Notice that the response code is in the 200 series, but not 200. In this case, it is 201 Created.

Tasks

With the basics covered, make the following REST calls to complete this practice.

1. Retrieve a list of all customers by making a GET call to: `http://localhost:8080/customers`
2. Using the PUT method, update customer 101 with the following JSON data. Hint: The syntax for update is almost exactly the same as add except the HTTP PUT method is used instead of POST.
   ```
   {"id":101, "firstName":"John
   Quincy","lastName":"Adams","email":"jqadams@example.com","city":"Braintree","s
   tate":"MA", "birthday":"1767-07-11"}
   ```
3. Delete customer 104.
   Hint: The syntax is almost exactly the same as a GET except the HTTP method DELETE is used instead.

Practice 16-4 (Solution)

Get All Customers

Response

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Fri, 26 Feb 2016 23:40:16 GMT

[{"id":100,"firstName":"George","lastName":"Washington","email":"gwash@example.
com","city":"Mt Vernon","state":"VA","birthday":"1732-02-
23"},{"id":101,"firstName":"John","lastName":"Adams","email":"jadams@example.co
m","city":"Braintree","state":"MA","birthday":"1735-10-
30"},{"id":102,"firstName":"Thomas","lastName":"Jefferson","email":"tjeff@examp
le.com","city":"CharlottesVille","state":"VA","birthday":"1743-04-
13"},{"id":103,"firstName":"James","lastName":"Madison","email":"jmad@example.c
om","city":"Orange","state":"VA","birthday":"1751-03-
16"},{"id":104,"firstName":"James","lastName":"Monroe","email":"jmo@example.com
","city":"New York","state":"NY","birthday":"1758-04-
28"},{"id":105,"firstName":"Abigail","lastName":"Adams","email":"aadams@example
.com","city":"Braintree","state":"MA","birthday":"1744-11-22"}]
```

Update Customer 101

Response

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Length: 0
Date: Fri, 26 Feb 2016 23:48:09 GMT
```

Delete Customer 104

Response

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Length: 0
Date: Fri, 26 Feb 2016 23:53:07 GMT
```

Get all Data After Operations

If you retrieve all the customers after these operations your data should look something like this. Note: If you have to restart your Spring Boot application any changes to data stored in the application are lost.

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Fri, 26 Feb 2016 23:55:44 GMT
```

[{"id":100,"firstName":"George","lastName":"Washington","email":"gwash@example.com","city":"Mt Vernon","state":"VA","birthday":"1732-02-23"},{"id":101,"firstName":"John Quincy","lastName":"Adams","email":"jqadams@example.com","city":"Braintree","state":"MA","birthday":"1767-07-11"},{"id":102,"firstName":"Thomas","lastName":"Jefferson","email":"tjeff@example.com","city":"CharlottesVille","state":"VA","birthday":"1743-04-13"},{"id":103,"firstName":"James","lastName":"Madison","email":"jmad@example.com","city":"Orange","state":"VA","birthday":"1751-03-16"},{"id":105,"firstName":"Abigail","lastName":"Adams","email":"aadams@example.com","city":"Braintree","state":"MA","birthday":"1744-11-22"}]

Practice 16-5: Testing a Rest Application

Overview

With your newly acquired REST testing knowledge, perform some REST operations on the sample Spring Boot application.

Note: For this practice, you will be creating JSON data. To make sure that the data you create is properly formatted, you can use a JSON validator like this one: http://jsonlint.com/.

Tasks

Perform the following tasks to complete this practice.

1. Stop the Spring Boot Application by hitting **Control + C** in the command prompt window.
2. Restart the sample application. (This will clear out any previous changes you have made to the customer data.)
3. Add yourself to the customer database with ID 106. This will require you to create a set of JSON data with your information. Include: First Name, Last Name, Email, City, State, and Birthday.
4. Delete record 101.
5. Update record 102 with the following information. You will need to convert the following information into JSON format.

    ID: 102
    First name: James
    Last name: Polk
    Email: jpolk@example.com
    City: Pineville
    State: North Carolina
    Birthday: 1833-03-04

When you have completed all the operations, retrieving the list of all customers from the REST application should produce output similar to the following:

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Sat, 27 Feb 2016 01:03:38 GMT

[{"id":100,"firstName":"George","lastName":"Washington","email":"gwash@example.
com","city":"Mt Vernon","state":"VA","birthday":"1732-02-
23"},{"id":102,"firstName":"James","lastName":"Polk","email":"jpolk@example.com
","city":"Pineville","state":"NC","birthday":"1833-03-
04"},{"id":103,"firstName":"James","lastName":"Madison","email":"jmad@example.c
om","city":"Orange","state":"VA","birthday":"1751-03-
16"},{"id":104,"firstName":"James","lastName":"Monroe","email":"jmo@example.com
","city":"New York","state":"NY","birthday":"1758-04-
28"},{"id":106,"firstName":"John","lastName":"Doe","email":"jdoe@example.com","
city":"Witchita","state":"KS","birthday":"1995-02-25"}]
```

Submit the script or a screenshot of the tool that you used to perform the operation. Also, provide the JSON text of all the customers after you have performed all the operations listed above.

Practice 16-5 (Solution)

See the solutions directory for this practice to see suggested cURL scripts.

Practice 17-1: Creating a Spring Boot REST Application (Part 1)

Overview

In this practice, set up and create the basic functions of a Spring Boot REST application.

Set Up Your Maven Spring Boot Project

The first step to creating the application is to set up the project. This consists of setting up the directory structure and the `pom.xml` file.

1. Navigate to the directory where you want to create the project. For example: `C:\labs\17spring`.
2. Create the following directory structure.

```
.
└──src
    ├──main
    │   ├──java
    │   │   └──com
    │   │       └──example
    │   │           └──rest
    │   └──resources
    └──test
        └──java
            └──com
                └──example
                    └──rest
```

3. At the root of the structure, create the `pom.xml`.
4. Copy the following configuration information into the file.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.springframework</groupId>
    <artifactId>17.1-hello-rs</artifactId>
    <version>17.1.0</version>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.3.3.RELEASE</version>
    </parent>

    <dependencies>
        <dependency>
```

```
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
    </dependencies>

    <properties>
        <java.version>1.8</java.version>
    </properties>


    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>

            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>exec-maven-plugin</artifactId>
                <version>1.4.0</version>
                <executions>
                    <execution>
                        <goals>
                            <goal>java</goal>
                        </goals>
                    </execution>
                </executions>
                <configuration>
                    <mainClass>com.example.rest.Application</mainClass>
                </configuration>
            </plugin>

        </plugins>
    </build>

    <repositories>
        <repository>
            <id>spring-releases</id>
            <url>https://repo.spring.io/libs-release</url>
        </repository>
    </repositories>
    <pluginRepositories>
        <pluginRepository>
            <id>spring-releases</id>
            <url>https://repo.spring.io/libs-release</url>
        </pluginRepository>
    </pluginRepositories>
</project>
```

Note: There are a couple of minor changes from the default `pom.xml` file provided on the Spring.io website. The Maven exec plug-in has been added to the file so that you can use `exec:java` to execute your application from the Maven command line. In addition, at the top of the file, the `artifactId` and `version` values have been updated to reflect the number of this practice.

5. Save the file.

6. Your project is now set up. However, there are no classes in the project. See directory `17-01A` for the Maven project set up so far.

Create a Simple Hello World Spring Boot Application

Next, set up an application class to launch the Spring Boot application.

1. Open the Maven project in NetBeans.
2. Create a new Java class named `Application` in the `com.example.rest` package.
3. Replace the empty class with the following code.

```
package com.example.rest;

import java.util.Properties;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {

    public static final Properties myProps = new Properties();

    public static void main(String[] args) {
        // Set properties

        myProps.setProperty("server.address", "localhost");
        myProps.setProperty("server.port", "8080");

        SpringApplication app = new SpringApplication(Application.class);
        app.setDefaultProperties(myProps);
        app.run(args);

    }
}
```

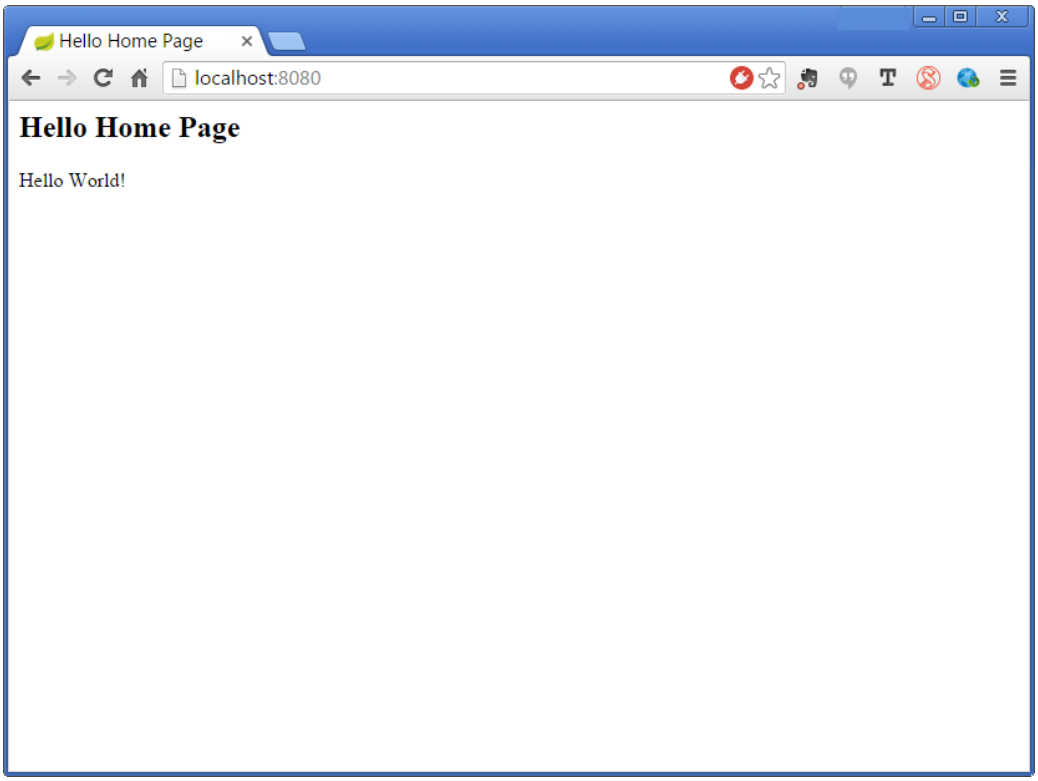
Note: This starts a Spring Boot application to listen locally on port 8080.

4. Save the file. Now you have a running Spring Boot application, but no content for it, because we have not yet added any REST controllers. However, you can add static webpages to a Spring Boot application for testing purposes.
5. In NetBeans, click the **Files** tab.
6. Navigate to the `src\main\resources` directory.
7. Create a subdirectory named `public`.
8. In the `public` directory, create an `index.html` file.
9. Put the following text into that file.

```
<html>
<head>
<title>Hello Home Page</title>
</head>
<body>
<h2>Hello Home Page</h2>
<p>Hello World!</p>
</body>
</html>
```

10. Save the file.

11. Compile your project: `mvn clean compile`
12. Run your project: `mvn exec:java`
13. Open a browser and connect to `http://localhost:8080`. You should see a window that looks like the following screenshot:



14. Your Spring Boot application is now running. See the `17-01B` directory to see the Maven project so far.

Set Up the REST Application Classes

Set up the Java classes needed by the REST application.

1. Create a `Customer` class to represent customer data. Use the following source code to create the class.

```
package com.example.rest;

public class Customer {
  private final long id;
  private final String firstName;
  private final String lastName;
  private final String email;
  private final String city;
  private final String state;
  private final String birthday;

  public Customer(){
      super();
      id = 0;
      firstName = "";
      lastName = "";
      email = "";
      city = "";
      state = "";
```

```
      birthday = "";
  }

  public Customer(long id, String firstName, String lastName, String email,
String city, String state, String birthday){
      this.id = id;
      this.firstName = firstName;
      this.lastName = lastName;
      this.email = email;
      this.city = city;
      this.state = state;
      this.birthday = birthday;
  }

  public long getId(){
    return this.id;
  }

  public String getFirstName() {
    return this.firstName;
  }

  public String getLastName() {
    return this.lastName;
  }

  public String getEmail(){
    return this.email;
  }

  public String getCity() {
    return this.city;
  }

  public String getState() {
    return this.state;
  }

  public String getBirthday(){
    return this.birthday;
  }

  @Override
  public String toString(){
    return "ID: " + id
        + " First: " + firstName
        + " Last: " + lastName + "\n"
        + "EMail: " + email + "\n"
        + "City: " + city
        + " State: " + state
        + " Birthday " + birthday;
  }

}
```

2. **Why use `final`?** In the practices for Lesson 18, your data is deployed to a Tomcat server, which is inherently multithreaded. That means multiple clients might try to change your data at the same time.

To prevent anything weird from happening, the data needs to be immutable. What does immutable mean? In English, immutable means read-only. By setting the variables to `final`, once initialized, the field values cannot be changed. Thus the data is read-only. Read-only data is inherently thread-safe.

3. Create a `MockCustomerList` class to store customer data in an array list. Use the following source code to create the class.

```
package com.example.rest;

import java.util.concurrent.CopyOnWriteArrayList;

public class MockCustomerList {
  private static final CopyOnWriteArrayList<Customer> cList = new
CopyOnWriteArrayList<>();

  static {
    // Create list of customers
    cList.add(
        new Customer(100, "George", "Washington", "gwash@example.com", "Mt
Vernon", "VA", "1732-02-23")
    );

    cList.add(
        new Customer(101, "John", "Adams", "jadams@example.com", "Braintree",
"MA", "1735-10-30")
    );

    cList.add(
        new Customer(102, "Thomas", "Jefferson", "tjeff@example.com",
"CharlottesVille", "VA", "1743-04-13")
    );

    cList.add(
        new Customer(103, "James", "Madison", "jmad@example.com", "Orange",
"VA", "1751-03-16")
    );

    cList.add(
        new Customer(104, "James", "Monroe", "jmo@example.com", "New York",
"NY", "1758-04-28")
    );

  }

  private MockCustomerList(){}

  public static CopyOnWriteArrayList<Customer> getInstance(){
    return cList;
  }

}
```

Why not `ArrayList`? `CopyOnWriteArrayList` is a thread-safe implementation for `ArrayList`. As stated before, because your application will run in a multi-threaded environment, a thread-safe data structure is required. The `CopyOnWriteArrayList` works just like the `ArrayList` that you learned about earlier in the course, except it is a thread-safe implementation.

4. Create a `NotFoundException` class that is used to handle exceptions. Use the following source code to create the class.

```
package com.example.rest;

public class NotFoundException extends RuntimeException {

    /*
     * Create a HTTP 404 (Not Found) exception.
     */
    public NotFoundException(String message) {
        super(message);
    }
}
```

5. Create a `JsonError` class to pass error messages to REST clients. Use the following source code to create the class.

```
package com.example.rest;

public class JsonError {
  private String type;
  private String message;

  public JsonError(String type, String message){
    this.type = type;
    this.message = message;
  }

  public String getType(){
    return this.type;
  }

  public String getMessage(){
    return this.message;
  }
}
```

6. Create the `CustomerController` class that you will use to create your REST application. Use the following source code to create the class.

```
package com.example.rest;

import java.util.concurrent.CopyOnWriteArrayList;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/customers")
public class CustomerController {

    private final CopyOnWriteArrayList<Customer> cList =
MockCustomerList.getInstance();

    // Code get all customers here

    // Code get a customer here


    // Throw this exception when a lookup fails
    @ExceptionHandler(NotFoundException.class)
    @ResponseBody
    public ResponseEntity<?> myError(Exception exception) {
        return new ResponseEntity<>(new JsonError("ID not found error:",
exception.getMessage()), HttpStatus.NOT_FOUND);
    }
}
```

Note: The `@RestController` and `@RequestMapping` annotations identify the class as RESTful. This way Spring Boot looks for appropriately annotated methods as request handlers for the application. Spring Boot uses the `ResponseEntity` class to set HTTP response codes. The class uses an unbounded generic <?> so that any class may be used with it.

7. Make sure that your application compiles without any errors before proceeding to the next section. To see a complete Maven project with the steps completed so far, see directory `17-01C`.

Add GET Methods to the REST Application

With the classes set up for your application, complete the following methods for the REST controller class. Then build and test your application.

1. Create a `GET` method handler that returns all the customers in the array list. The list of all customers should be returned when a `GET` request is made to the following URL:
   `http://localhost:8080/customers`

2. Create a `GET` method handler that returns a single customer if the customer ID is specified in the URL. Return a single customer when a `GET` request is made to the following example URL: `http://localhost:8080/customers/100`.

   If the customer does not exist, throw a not found exception and return an appropriate error message.
3. Compile, deploy, and test your application. You can use your browser, cURL or Postman for testing. See directory `17-01` to see a completed Maven project solution for this practice.

Practice 17-1 (Solution)

See the completed source files for the Maven project in the `17-01-sol` directory.

Practice 17-2: Creating a Spring Boot REST Application (Part 2)

Overview

In this practice, you complete the remaining methods in the Spring Boot REST application.

Complete the Spring Boot REST Application

1. Create a `POST` method handler that adds a customer to the array list. The method should respond with a response code of 201 Created. The new customer data should be submitted in a JSON format to the following URL:
   `http://localhost:8080/customers`
2. Create a `PUT` method handler that updates a customer in the array list identified by ID. The method should respond with a response code of 200 OK if the operation is successful. If the lookup of the ID fails, respond with a 404 Not Found response code. The updated customer data should be submitted in JSON format to a URL similar to the following:
   `http://localhost:8080/customers/101`
3. Create a `DELETE` method handler that deletes a customer in the array list identified by ID. The method should respond with a response code of 200 OK if the operation is successful. If the lookup of the ID fails, respond with a 404 Not Found response code. The updated customer data should be submitted in JSON format to a URL similar to the following:
   `http://localhost:8080/customers/101`
4. Compile, deploy, and test your application. You can use cURL or Postman for testing. See directory `17-02` to see a completed Maven project solution for this practice.

Practice 17-2 (Solution)

For the completed Maven project, see directory `17-02-sol`.

Practice 17-3: Creating a Soccer Data Model

Overview

In this practice, you create a `Player` class to represent soccer player stats. You will use an array list to store player classes as your data model in the Practice 17-4.

Create an Empty Maven Project

Create an empty Maven project to store the code for this practice.

1. Use the Maven `quickstart` archetype to create an empty project. Use the following command:
   ```
   mvn archetype:generate -DgroupId=com.example.rest -DartifactId=PlayerProj -
       DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
   ```
2. Open the project in NetBeans and verify that the com.example.rest package has been created with the `App.java` class.
3. Before you can run the project, you need to add some additional configuration for Maven.
   a. Set the Java version and encoding:

```
<properties>
    <java.version>1.8</java.version>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
```

   b. Add the build section to set the compiler, execution goal, and package goal. This allows you to run the project with the `exec:java` command and create a package where the `Main` method is properly set.

```
<build>
    <plugins>
        <!-- Set a JDK compiler level -->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.3.2</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>

        <plugin>
            <groupId>org.codehaus.mojo</groupId>
            <artifactId>exec-maven-plugin</artifactId>
            <version>1.4.0</version>
            <executions>
                <execution>
                    <goals>
                        <goal>exec</goal>
                    </goals>
                </execution>
            </executions>
            <configuration>
                <mainClass>com.example.rest.App</mainClass>
            </configuration>
        </plugin>

        <!-- Make this jar executable -->
```

```
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-jar-plugin</artifactId>
                <version>2.6</version>
                <configuration>
                    <archive>
                        <manifest>
                            <mainClass>com.example.rest.App</mainClass>
                        </manifest>
                    </archive>
                </configuration>
            </plugin>

        </plugins>
    </build>
```

4. Save the changes.
5. Run the application using `mvn exec:java`. The application should print "Hello World!".
6. Package the application with: `mvn package`.
7. Now execute the application with `java -jar target\PlayerProj-1.0-SNAPSHOT.jar`. The output should be the same.

   Note: If you change the configuration, the output JAR file name may differ from this example.

Create the Player Class

Next, create the Player class to store data about each player.

1. The player class fields have the following signatures.

```
  private final long id;
  private final String team;
  private final String name;
  private final String position;
  private final String number;
  private final String country;
  private final long goals;
  private final long yellowCards;
```

   Why use `final`? In the practices for Lesson 18, your data is deployed to a Tomcat server, which is inherently multithreaded. That means multiple clients might try to change your data at the same time. To prevent anything weird from happening, the data needs to be immutable. What does immutable mean? In English, immutable means read-only. By setting the variables to `final`, once initialized, the field values cannot be changed. Thus the data is read-only. Read-only data is inherently thread-safe.

2. Create a zero argument constructor for the class. The constructor should take no arguments and initialize all the variables.

   Note: Some frameworks require a zero argument because they use a technique called reflection to make inferences about a class. Some of the frameworks included with Spring Boot have this requirement.
3. Create a constructor with the following signature. Initialize the player object with the data passed into the constructor.

```
  public Player(long id, String team, String name, String position, String
number, String country, long goals, long yellowCards)
```

4. Create a `get` method for each field in the class.

5. Save the class.

Create the Player List

Create a `MockPlayerList` class that will store player data. The player data is included below.

1. Create a field for your array list called `pList`. Use the `CopyOnWriteArrayList` as the class type of your list.

```
  private static final CopyOnWriteArrayList<Player> pList = new
CopyOnWriteArrayList<>();
```

   Why not `ArrayList`? `CopyOnWriteArrayList` is a thread-safe implementation for `ArrayList`. As stated before, your application will run in a multi-threaded environment so a thread-safe data structure is required. The `CopyOnWriteArrayList` works just like the `ArrayList` you learned about earlier in the course.

2. Initialize your array list in a static block using the following data. The field names in order are: `id`, `team`, `name`, `position`, `number`, `country`, `goals`, `yellowCards`.

```
1, "Pelicans", "Dorothy Parker", "F", "9", "USA", 9, 2
2, "Pelicans", "James Joyce", "M", "7", "Ireland", 8, 0
3, "Magpies", "Emma Orczy", "F", "10", "Hungary", 8, 1
4, "Hawks", "Charles Dickens", "F", "10", "England", 7, 1
5, "Robins", "JRR Tolkien", "F", "9", "England", 7, 0
6, "Pelicans", "William Makepeace", "F", "10", "England", 6, 1
7, "Magpies", "Sean O'Casey", "M", "8", "Ireland", 6, 0
8, "Hawks", "James Fenimore Cooper", "D", "2", "USA", 5, 0
9, "Hawks", "Alexandre Dumas", "M", "6", "France", 5, 2
```

3. Create a `getInstance` method to return your array list. Note that this is a `static` method. Using the `static` keyword ensures that only one copy of this array list is available to other classes.

```
  public static CopyOnWriteArrayList<Player> getInstance(){
    return pList;
  }
```

4. Create a private zero argument constructor that is empty. This prevents any other classes from creating an instance of this class. The class is designed to only create a `CopyOnWriteArrayList` and return that to other classes.
5. Save your class.

Test Your Class

You are now ready to test your data model.

1. Rewrite the App.java class to create a `CopyOnWriteArrayList` array list and print its contents to the console.
2. Run your application using: `mvn exec:java`.
3. Compile and package your application.
4. Execute your application using: `java -jar`
5. Your output should look similar to the following:

```
ID: 1 Team: Pelicans Name: Dorothy Parker Position: F Number: 9 Country: USA
Goals: 9 Yellow Cards: 2
ID: 2 Team: Pelicans Name: James Joyce Position: M Number: 7 Country: Ireland
Goals: 8 Yellow Cards: 0
```

```
ID: 3 Team: Magpies Name: Emma Orczy Position: F Number: 10 Country: Hungary
Goals: 8 Yellow Cards: 1
...
```

Practice 17-3: Create Soccer Data Model (Solution)

See the completed Maven project for the source code for this practice in directory `17-03-sol`.

Practice 17-4: Creating a Spring Boot REST Application

Overview

Our client has a new request. They're asking us to create a micro REST web service for their league that will keep track of player statistics. I want you on this project. I'm sure you can fit it into your schedule.

Sometime in the future, we will add additional web services to cover other business aspects of the league. But for now, just worry about writing a Spring Boot REST program that manages player stats. Good luck!

Tasks

Create a new Maven Spring Boot REST project. Use the Player classes you created in Practice 17-3 as your data model.

Your application should include the following classes.

- `Player` - A class that represents each player and their statistics.
- `MockPlayerList` - A class that creates a list of player objects.
- `Application` - A Spring Boot application launcher class. Set the host name to `localhost` with the network port set to `8080`.
- `PlayerController` - The Spring Boot REST controller class. Put all your methods to handle web service requests here.
- `NotFoundException` - An exception class to handle not found errors.
- `JsonError` - An error class that can create messages to return in a REST response.

Your REST application should provide the following operations.

- `/players/`
  - `GET` - Returns all the players in the database in a JSON format.
  - `POST` - Add a player using JSON data provided in the request body.
- `/players/{id}`
  - `GET` - Return a single player identified by the integer Id. If no player exists with that number, throw an `NotFoundException`.
  - `PUT` - Update a single player identified by the integer Id. If no matching Id is found, return an error message.
  - `DELETE` - Delete a single player identified by the integer Id. If no matching Id is found, return an error message.
- `/players/country/{name}` - Return all the players from the country name specified. If no players are found, throw a `NotFoundException`.
- `/players/goals/{count}` - Return all the players that have a matching or higher goal total. If no players are found, throw a `NotFoundException`.

To test your application, perform the following operations.

- Add yourself as player 10. Give yourself a player number of 8 with 10 goals and 2 yellow cards. Set your country to France, your team to the Hawks, and your position to defender (D).
- Delete the player with an ID of 4.

- Update the player with an ID of 2 with the following information:
  `{"id":2,"team":"Pelicans","name":"Jim Bob Joyce","position":"M","number":"7","country":"Ireland","goals":10,"yellowCards":2}`
- Get a JSON response listing all the players from France.
- Get a JSON response with all players who have scored eight or more goals.

When you are done, retrieve all the players in the database. Your output should look similar to the following.

```
[{"id":1,"team":"Pelicans","name":"Dorothy Parker","position":"F","number":"9","country":"USA","goals":9,"yellowCards":2},{"id":2,"team":"Pelicans","name":"Jim Bob Joyce","position":"M","number":"7","country":"Ireland","goals":10,"yellowCards":2},{"id":3,"team":"Magpies","name":"Emma Orczy","position":"F","number":"10","country":"Hungary","goals":8,"yellowCards":1},{"id":5,"team":"Robins","name":"JRR Tolkien","position":"F","number":"9","country":"England","goals":7,"yellowCards":0},{"id":6,"team":"Pelicans","name":"William Makepeace","position":"F","number":"10","country":"England","goals":6,"yellowCards":1},{"id":7,"team":"Magpies","name":"Sean O'Casey","position":"M","number":"8","country":"Ireland","goals":6,"yellowCards":0},{"id":8,"team":"Hawks","name":"James Fenimore Cooper","position":"D","number":"2","country":"USA","goals":5,"yellowCards":0},{"id":9,"team":"Hawks","name":"Alexandre Dumas","position":"M","number":"6","country":"France","goals":5,"yellowCards":2},{"id":10,"team":"Hawks","name":"Victor Hugo","position":"D","number":"8","country":"France","goals":10,"yellowCards":2}]
```

Practice 17-4: Create Soccer Data Model (Solution)

See the completed Maven project for the source code for this practice in directory `17-04-sol`.

Practice 18-1: Create an Application Archive with Maven

Overview

In this practice, you create an application archive that can be deployed to Oracle Application Container Cloud Service.

Create a New Project for This Lesson

Create a new project and rename it.

1. Change into the `cloud` directory.
2. Create a new directory for lesson 18: `mkdir 18`
3. In the new directory, copy the solution for practice `17-02` into an `18-01` directory.
4. Update the `pom.xml` file for the 18-01 project name and location.
5. Compile and package the application.
6. Test running the application and application JAR. Correct any errors you encounter.

Note: Commit your changes to Git as you proceed through the updates.

Update the Project for OACCS Deployment

Update your application so that it can start using environment variables set by the operating system.

1. Edit the `Application.java` class.
2. Add code to read the `HOSTNAME` environment variable. If the value is set, use that value to start your Spring Boot application. If the value is not set, use `localhost`.
3. Add code to read the `PORT` environment variable. If the value is set, use that value to start your Spring Boot application. If the value is not set, use `8080`.
4. Compile and package your application. Correct any errors.
5. Test the application using the JAR file to make sure it works.
6. Set the `PORT` environment variable to `8088`.
7. Run the application and retest. Make sure your application is now listening on port `8088`.

Update the Project to Produce an Application Archive

In this section, update the project files so that an application archive is created that is ready for deployment on the OACCS.

1. The modifications that follow to the 18-01 project you have been working on.
2. First, add a `manifest.json` file to your project. You can use the following sample JSON text for your file.

```
{
    "runtime": {
        "majorVersion": "8"
    },
    "command": "java -jar 18-01-customer-rs-18.1.0.jar",
    "release": {
        "version": "18.1.0",
```

```
        "build": "24",
        "commit": "1A2B345"
    },
    "notes": "Customer Spring Boot Web Service"
}
```

3. Save the file in the root directory of your project.
4. Next create a `bin.xml` file to specify how the assembly plugin will build your application archive. The following is sample text for that file:

```
<assembly xmlns="http://maven.apache.org/plugins/maven-assembly-
plugin/assembly/1.1.3"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/plugins/maven-assembly-
plugin/assembly/1.1.3 http://maven.apache.org/xsd/assembly-1.1.3.xsd">
    <id>distribution</id>
    <formats>
        <format>zip</format>
        <format>tar.gz</format>
    </formats>
    <includeBaseDirectory>false</includeBaseDirectory>
    <files>
        <file>
            <source>manifest.json</source>
            <outputDirectory></outputDirectory>
        </file>
    </files>
    <fileSets>
        <fileSet>
            <directory>${project.build.directory}</directory>
            <outputDirectory></outputDirectory>
            <includes>
                <include>18-01-customer-rs-18.1.0.jar</include>
            </includes>
        </fileSet>
    </fileSets>
</assembly>
```

5. Save the file in the root directory of the project.
6. Update the `pom.xml` file to include configuration for the assembly plugin. You can use the following XML code:

```
            <plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <version>2.6</version>
                <configuration>
                    <descriptors>
                        <descriptor>bin.xml</descriptor>
                    </descriptors>
                    <finalName>${project.build.finalName}-dist</finalName>
                    <appendAssemblyId>false</appendAssemblyId>
               </configuration>
                <executions>
                    <execution>
                        <id>make-assembly</id>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
```

```
                </executions>
            </plugin>
```

7. Save the updated `pom.xml` file.
8. Perform a Maven `clean` and `package`.
9. Examine the `target` directory. You will notice that a `.zip` and a `.tar.gz` distribution file have been generated. These are application archive files that you can use to deploy to OACCS.
10. Copy one of the files to a temporary directory and unzip it. Notice that the file contains your application's JAR file and the `manifest.json` file.
11. You are now ready to deploy your application.

Practice 18-1 (Solution)

See the solutions directory for the project files for this practice.

Practice 18-2: Complete a Spring Boot REST Web Service

Overview

In this practice, deploy your newly updated project to OACCS. The steps for deployment are outlined in the lecture, with screenshots. They list of steps are included here for your convenience.

Tasks

1. Log in to your Oracle Cloud account.
2. Open the **Service Console** for Oracle Application Container Cloud Service.
3. From the Application List page, click **Create Application**.
4. Fill out the Create Application dialog box and select your application archive from your local file system.
5. Click **Create**. Your application is deployed in a few minutes.
6. After your application deploys, explore the application tabs that are available to you. Tabs include:
    - Overview
    - Deployments
    - Administration
    - Logs and Recordings
7. You are now ready to test your application.

Practice 18-2 (Solution)

There are no solution files for this practice.

Practice 18-3: Testing the Application on OACCS

Overview

In this practice, test the deployment of your application on OACCS.

Tasks

Test the deployment of your application on OACCS.

1. Get the URL that your application is listening on.
2. Perform each of the following tests on your system.
    a. Get the JSON data for individual customer objects.
    b. Get the JSON data for all customer objects.
    c. Add a new customer to the customer list.
    d. Update a customer in the customer list.
    e. Delete a customer from the customer list.
3. Validate that all of the operations are functioning in the cloud.

Note: Your Maven project directory includes a cURL directory with sample scripts for performing all the test listed.

Practice 18-3 (Solution)

There are no solution files for this practice.

Practice 18-4: Scaling the Application

Overview

In this practice, scale your application out.

1. Select your application from the OACCS application list.
2. Scale your application out to two instances.
3. After the scaling is complete, retest your application.
    a. If you make changes to the customer list, notice if those changes change between calls.
    b. The load balancer will choose which instance responds on every call to the application.
    c. Because you have two instances running, each will have a different data set. This is why in a production system you would need to store your data in a shared database.
4. After you have completed your testing, scale your application back in to one instance.

Practice 18-4 (Solution)

There are no solution files for this practice.

Practice 18-5: Deploying Your Application to OACCS

Overview

Now that you have created a Spring Boot REST application, it is time to update the soccer application for deployment on the Oracle Application Container Cloud Service. Make the required changes and deploy your application.

Tasks

It is now time to update the soccer application so that it can be deployed on the OACCS.

1. Start with the project that you completed in Practice 17-4.
2. Update the application so that is can read the PORT and HOSTNAME environment variables.
3. Update the Maven pom.xml file so that it generates a deployable OACCS application archive.
4. Build and package for application archive.
5. Deploy your application to the OACCS.
6. Test your application on the OACCS.

Practice 18-5 (Solution)

See the solutions directory for the project files for this practice.