Name: \_\_\_\_\_\_Jihad Alsahori\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |
| --- |
| **ASSIGNMENT GOALS:**   1. **Improve a previously developed solution with an eye toward “refactoring” and “configurability”.** 2. **Gain additional experience with some select Core Java Classes** 3. **Implement a Custom Exception Class** |

NOTE: package up the programming assignment Classes under:**edu.cuny.csi.csc330.lab6**

**Reading assignment:**“Starting out with Java” Textbook pp 701~732 (Exception Handling)“Pragmatic Programmer” Textbookpp 112 (Dead Programs tell no Lies) ~ bottom of 117

**Assignment: Making our** LottoQuickPicker**Solution much more “flexible” (100 pts)**

* **FocusCustomExceptions and Configurability.**

This programming assignment revolves around embellishing the design and implementation of Lab 3. Our requirement for Lab 3 drove us to implement a Class called LottoQuickPickerthat represented the popular NYS lotto game of the same name. Your solution leveraged the services of another Class – Randomizer – and was hopefully as modular as possible, but our solution was intended to be fairly fixed.That is, it modeled the rules of only 1 game.

|  |
| --- |
|  |

Imagine that NYS was very thrilled with your LottoQuickPickerimplementation, and offered you a chance to build a “Quick Picker” Solution for all games – both present and future. How would you go about re-factoring your LottoQuickPickersolution into one that covered our new requirements?

One possible approach might look like this.

|  |
| --- |
|  |

That is, you might have an abstract “base Class” of sorts that allows you to share common data and methods with game specific sub-classes.

This is a reasonable approach, but can it be better - more flexible and easier to maintain?

First, what ongoing changes would you be responsible for?

1. New game implementations.
2. Rule modifications to existing games.

**NOTE: all games force our implementations to include 2 pools of numbers rather than just one – as many NYS lotto games pick N numbers from a primary pool and N “supplementary numbers”.**

What if we could build a single Class solution that would support ALL games – present and future? - one that made use of “configuration files” (property files) to drive the specific of each game.

What would such a solution look like?

|  |
| --- |
|  |

The model above could rely on a collection of Java properties files. Each properties file would dictate the rules of one Lotto Game.

A properties file with the following format hasall information required to cover all game behavior details.

Example 1

|  |
| --- |
| ###################################################  ## Pick10 game definition - pick 10 in 80 numbers  GameName=Pick10!  Pool1=10/80  Pool2=0/0  Vendor=CSC330CornerStore |

Example 2

|  |
| --- |
| ###################################################  ## Mega Million game definition - pick 5 in 75 and then 1 in 15  GameName=MegaMillions!  Pool1=5/75  Pool2=1/15  Vendor=CSC330CornerStore |

**Your assignment**:

1. Implement a solution that would NOT require us to modify any Java code in order to support an evolving Lotto Game set. One that supports the design suggested above – where:

* A new game would require you to add a new .properties file to your deployment.
* Updated rules to an existing game (or a Name Change) would require you to modify an existing .properties file.

And thus, unless something very fundamental changes about the way NYS Lotto works, you would never need to modify, recompile and re-test your code base.

**This can clearly be called a “Configurable” solution!**

1. Create a custom Exception called QuickPickerException that will be throw by your QuickPicker Class if one of the conditions occur.
   1. The specified game does not exist (missing .properties file)
   2. A required property is missing from thespecified .properties file.

You need to implement the following Classes:

1. **QuickPicker** – implements the behavior of a generic QuickPicker solution that depends on the details found in a game specific .propertiesfile – everything needed to provide the same functionality that your LottoQuickPickersolution provided.
2. **QuickPickerException**– needs to extend the Exception Class. Use the edu.cuny.csi.csc330.newradio.RadioExceptionClass as a solid point of reference / template.
3. Program Arguments: Your QuickPicker Class will require 1 argument and accept an second **optional argument**:
   * + 1. The name of the Game to be played. Name must match the name of the properties file – e.g., a file called MegaMillions.properties must exist to support a 1st argument of “MegaMillions” …
       2. The number of “bets/games” to be generated. Like your LottoQuickPickersolution, the default number of games should be **1**.
4. Your .properties files should live under the “src” folder/directory of your Eclipse project folder.

You will be utilizing the java.util.PropertyResourceBundle Class to gather game properties details.

Refer the edu.cuny.csi.csc330.examples.PropertyBundleDemo Class in our source tree.

**EXTRA CREDIT FEATURE +5 points**

|  |
| --- |
| ---------------------------------  -------- Game Name --------  Sun Jan 27 19:56:38 EST 2019  ( 1) 02 06 13 15 22 33  ( 2) 10 11 13 18 32 38  ( 3) 06 08 16 19 29 34  ( 4) 09 22 26 33 36 37  **Odds of Winning: 1 in n,nnn,nnn<<<<<< EXTRA CREDIT INFO**  ----- (c) S.I. Corner Deli -----  --------------------------------- |

* In Lab 3 we asked, how would you calculate the odds of winning the LottoQuickPicker Game? Well, here we go again – if you implemented this for the LottoQuickPickerGame, you need to genericize your solution is 2 ways:
* NOTE:
  + Your solution should work for any game specification
  + This time, you need to consider the odds of both theprimary and secondary number pools (that is, when a game has a secondary pool).

**You will be submitting**

* TwoJava source files:

1. QuickPicker.java
2. QuickPickerException.java

* One or more .properties files that drive the rules of one or more Game of your choice (an actual NYS Lotto game, or one you made up)

NOTE: at least one of your gamesmust use the second / supplementary pool of numbers (see game examples above).

* This Word Doc with the output section below filled in. Again, make sure you show a “sunny day” successful output as well as a failed attempt that throws a QuickPickerException as shown below.

\

**Sunny Day Scenario**

|  |
| --- |
| -----------------------------------  -------- <<Game Name>> --------  Thu Mar 31 08:58:17 EDT 2016  ( 1) 03 27 32 49 56 (( 03 ))  ( 2) 16 23 38 46 56 (( 04 ))  ( 3) 06 22 27 41 58 (( 04 ))  ( 4) 10 27 31 33 34 (( 02 ))  ( 5) 11 13 17 35 43 (( 01 ))  Odds of Winning: 1 in <<n,nnn,nnn>>  ----- (c) CSC330 Corner Store -----  ----------------------------------- |

**Rainy Day Scenario**

|  |
| --- |
| QuickPickerException [code=1, toString()=edu.cuny.csi.csc330.prof.lab6.QuickPickerException: Can't find: CashForLife]  Unknown Game - Can't locate specified game. |

**Place your program’s output here – for both a successful launch and one that threw aQuickPickerException:**

|  |
| --- |
|  |