Name: Michael Lewis

BU ID: U14985193

Homework 1

1. Define the following terms (as applied to Java)
   1. Lambda – Lambda expressions introduce a functional programming style into Java and are an anonymous collection of executable Java code that are similar to anonymous functions. Additionally, lambdas allow code to be written inline as literals and use type inference (as opposed to strict naming rules of Java). Despite the similarity to anonymous functions, however, lambdas represent an object of a specific target type that must be an interface and only have one nondefault method. For example, map(), filter(), and reduce() are prominent functional idioms offered through lambdas.
   2. Static – Static is a modifier that specifies the fields and methods it is associated with as class members rather than instance members. Therefore, static methods do not have an implicit “this” object reference and are instead invoked through the containing class name. Additionally, static member types can only access other static members of its containing class and type.
   3. Volatile – Volatile is a modifier that indicates the field may not be cached by a thread and must instead by read from and flushed to main memory. This is one of Java’s keywords that helps the programmer deal with concurrent access to data. Effectively, the volatile modifier means that the value of the field or variable must be reread from main memory.
   4. Interface – Interfaces are Java’s solution to the limitations of single inheritance. Instead of allowing multiple inheritance and running into the “deadly diamond of death” problem, Java implements the interface reference type to act as an API. Effectively, it provides a description of the type and methods that an implementing class must provide (unless the interface method is marked as default in which case the implementation is provided by the interface). Furthermore, an implementing class can implement multiple interfaces unlike single class inheritance.
   5. Abstract – Abstract is another modifier keyword. An abstract class cannot be instantiated, and an abstract method is simply a method with no body (e.g. a signature only). The concept of “abstract” is linked to inferface because all interfaces are abstract. Effectively, an abstract class can act as a superclass that provides some characteristics that the subclass must implement.

2.

**Source Code - Implementation:**

*/\*\*  
 \* Guessing Game:  
 \* A class that implements the "I am thinking of a number between 1 and 1000". The computer will think of the number,  
 \* and the user will repeatedly guess until he/she answers correctly. The program will tell the user if the guesses  
 \* are too high or too low. This only needs to be a console game. When the user guesses the correct number, the game  
 \* should show the user how many turns it took. You must handle incorrect input (letters when numbers were wanted).  
 \*  
 \** ***@author*** *: Michael Lewis  
 \* Date : 9/14/18  
 \* Class : MET CS342  
 \* Issues : None known  
 \*/*public class GuessingGame {  
 private boolean hasWon = false; // Track whether or not a user guessed the target number  
 private int numberOfGuesses = 0; // Track how many guesses it took to find the target number  
 private int targetNumber; // The number the computer picks and the user tries to guess  
  
 */\*\*  
 \* public GuessingGame()  
 \* Construct a Game with a specified target number  
 \*/* public GuessingGame() {  
 setTargetNumber();  
 }  
  
 */\*\*  
 \* private void setHasWon(boolean hasWon)  
 \* Set the game status equal to the inverse of the given parameter. True become False and False become True  
 \** ***@param*** *hasWon The current status of the game  
 \*/* public void setHasWon(boolean hasWon) {  
 this.hasWon = !hasWon;  
 }  
  
 */\*\*  
 \* private boolean getHasWon()  
 \* Get the status of the game. True if the user has guessed the target number. False otherwise  
 \** ***@return*** *True if the user has guessed the target number. False otherwise  
 \*/* public boolean getHasWon() {  
 return hasWon;  
 }  
  
 */\*\*  
 \* private void setNumberOfGuesses()  
 \* Increases the number of guesses by one  
 \* Precondition : The user has attempted a guess  
 \*/* public void setNumberOfGuesses() {  
 numberOfGuesses++;  
 }  
  
 */\*\*  
 \* private int getNumberOfGuesses()  
 \* Get the number of attempts the user has taken  
 \** ***@return*** *An integer value totaling the number of attempts the user has taken  
 \*/* public int getNumberOfGuesses() {  
 return numberOfGuesses;  
 }  
  
 */\*\*  
 \* private void setTargetNumber()  
 \* Set target number to a pseudo random number between 1 and 1000  
 \*/* public void setTargetNumber() {  
 this.targetNumber = (int) (Math.*random*() \* 1000) + 1;  
 }  
  
 */\*\*  
 \* private int getTargetNumber()  
 \* Get the target number  
 \* Precondition : Target number is a number  
 \** ***@return*** *A number between 1 and 1000  
 \*/* public int getTargetNumber() {  
 return targetNumber;  
 }  
}

**Source Code - Main method:**

import java.util.Scanner;  
  
*/\*\*  
 \* Play Game:  
 \* Starts the game play for the "I am thinking of a number between 1 and 1000" game.  
 \*  
 \** ***@author*** *: Michael Lewis  
 \* Date : 9/14/18  
 \* Class : MET CS342  
 \* Issues : None known  
 \*/*public class PlayGame {  
 */\*\*  
 \* public static void main(String[] args)  
 \* Prompt the user to enter a number. Require an additional entry if previous entry was not a number. Compare the  
 \* number to the target number. If the numbers match, print the number of guesses. Otherwise, prompt the user for  
 \* another entry.  
 \** ***@param*** *args The string argument is not used in this implementation  
 \* Precondition : User must enter an integer  
 \*/* public static void main(String[] args) {  
 GuessingGame game = new GuessingGame();  
  
 Scanner scanner = new Scanner(System.*in*);  
 while (!game.getHasWon()) { // Continue the game until the user enters the target number  
 System.*out*.println("Please guess an integer (e.g 500, 750): ");  
  
 // Require the user to enter an number, otherwise catch a NumberFormatException  
 // Print clues on where the target number lies relative to the previous entry  
 try{  
 game.setNumberOfGuesses();  
 int userGuess = Integer.*parseInt*(scanner.next().trim());  
 if (userGuess == game.getTargetNumber()) game.setHasWon(game.getHasWon());  
 else if (userGuess > game.getTargetNumber()) System.*out*.println("CLUE: " + userGuess + " is too large!");  
 else System.*out*.println("CLUE: " + userGuess + " is too small!");  
 } catch (NumberFormatException e) {  
 System.*out*.println("ERROR: You must guess an integer. Please try again.");  
 }  
 }  
  
 // The user entered the target number. Print the results and end the game  
 System.*out*.println("You won! It took you " + game.getNumberOfGuesses() + " guesses.");  
 scanner.close();  
 }  
}

**Output from first run:**

Please guess an integer (e.g 500, 750):

9.0

ERROR: You must guess an integer. Please try again.

Please guess an integer (e.g 500, 750):

500

CLUE: 500 is too large!

Please guess an integer (e.g 500, 750):

250

CLUE: 250 is too large!

Please guess an integer (e.g 500, 750):

175

CLUE: 175 is too large!

Please guess an integer (e.g 500, 750):

90

CLUE: 90 is too large!

Please guess an integer (e.g 500, 750):

50

CLUE: 50 is too small!

Please guess an integer (e.g 500, 750):

75

CLUE: 75 is too small!

Please guess an integer (e.g 500, 750):

85

CLUE: 85 is too large!

Please guess an integer (e.g 500, 750):

80

CLUE: 80 is too small!

Please guess an integer (e.g 500, 750):

84

You won! It took you 10 guesses.

**Output from run 2:**

Please guess an integer (e.g 500, 750):

500

CLUE: 500 is too large!

Please guess an integer (e.g 500, 750):

h

ERROR: You must guess an integer. Please try again.

Please guess an integer (e.g 500, 750):

250

CLUE: 250 is too small!

Please guess an integer (e.g 500, 750):

375

CLUE: 375 is too large!

Please guess an integer (e.g 500, 750):

300

CLUE: 300 is too small!

Please guess an integer (e.g 500, 750):

325

CLUE: 325 is too small!

Please guess an integer (e.g 500, 750):

350

CLUE: 350 is too large!

Please guess an integer (e.g 500, 750):

340

CLUE: 340 is too large!

Please guess an integer (e.g 500, 750):

335

CLUE: 335 is too large!

Please guess an integer (e.g 500, 750):

330

CLUE: 330 is too large!

Please guess an integer (e.g 500, 750):

327

CLUE: 327 is too small!

Please guess an integer (e.g 500, 750):

328

CLUE: 328 is too small!

Please guess an integer (e.g 500, 750):

329

You won! It took you 13 guesses.