COM S 227: Object-Oriented Programming
Exam 1 Practice
Spring 2024

DO NOT OPEN THIS EXAM UNTIL INSTRUCTED TO DO SO

Student Name: _____

ISU NetID (username): _____**@iastate.edu**

Instructions:
Closed book/notes, no electronic devices, no headphones. Time limit 75 minutes. Partial credit may be given for partially correct solutions. *If you have questions, please ask!*

- Use correct Java syntax for writing code. You do not need `import` statements.
- You are not required to write comments for your code; however, brief comments may help make your intention clear in case your code is incorrect.
- Use the abbreviation "SOP" for System.out.println
- It is ok to use literal values; you don't need to define constants for numbers.
- There are some excepts from the Java API documentation on the last page
- If you remove any of the last pages, you must turn them in with your exam

Question	Points
1	/14
2	/14
3	/30
4	/14
5	/14
6	/14
Total	/100

1. (14 pts) Complete the right-hand side of each assignment by writing an expression satisfying the stated requirement. (In some cases there is more than one correct answer. *Do not write any additional lines of code, just write an expression.*) The first one is done for you as an example

(EXAMPLE) A boolean value indicating whether the int variable **x** is an odd number

```
boolean isOdd = (x % 2 != 0);
```

(a) Given an int named **eggCount**, the number of full cartons of 12 eggs you can make with that many eggs

```
int fullCartons =
```

(b) Given an int named **eggCount**, the number of eggs left over after putting them into full cartons of 12 eggs

```
int leftOver =
```

(c) Given a String **str**, the last character of the string

```
char lastChar =
```

(d) A boolean value indicating whether the Strings **s1** and **s2** contain the same text

```
boolean areTheSameString =
```

(e) A boolean value indicating that the string **s** contains at least four characters.

```
boolean hasAtLeastFour =
```

(f) Given an integer number **d** of dollars and **c** of cents, a double that is the dollar value of **d** dollars and **c** cents

```
double totalValue =
```

(g) An integer containing the whole number part of a double **d**

```
int i =
```

2. (14 pts) *The class **Mystery** is defined as follows:*

```
public class Mystery
{
    private int leaf;
    public Mystery(int leaf)
    {
        this.leaf = leaf;
    }

    public int brew(int green) {
        if (green > 0) {
            if (green < 10) {
                green = green + leaf;
            }
        } else {
            green = 0;
        }
        if (green < 20) {
            leaf = leaf * 2;
        }
        return green;
    }
}
```

Below is a class ***MysteryTest*** with a main method, which constructs an instance of the class ***Mystery***, also shown below. What are the four lines of console output when the main method is executed? Write **ONLY** the output.

```
public class MysteryTest
{
    public static void main(String[] args)
    {
        Mystery m = new Mystery(5);
        System.out.println(m.brew(20)); // line 1
        System.out.println(m.brew(-9)); // line 2
        System.out.println(m.brew(5)); // line 3
        System.out.println(m.brew(5)); // line 4
    }
}
```

	Your Answer
Line 1	
Line 2	
Line 3	
Line 4	

3. (30 pts) ***THIS QUESTION HAS TWO PARTS. Part (a) is to write some simple test code, and part (b) is the complete class implementation, described in detail below.***

Consider a class **ParkingMeter** that models a coin-operated parking meter. We will assume it only takes quarters. Its public interface includes:

- A constructor **ParkingMeter(int minutesPerQuarter, int maximumTime)**, where *minutesPerQuarter* is the number of minutes you get for a quarter, and *maximumTime* is the maximum number of minutes. A newly created *ParkingMeter* has no time on it.
- A method **insertCoin(int howMany)** that simulates adding the given number of quarters. Inserting coins increases the time remaining on the meter, but not more than the maximum time.
- A method **getTimeRemaining()** that returns the time remaining on the meter in minutes as an int value.
- A method **passTime(int minutes)** that simulates the passage of time, that is, reduces the time remaining (but not below zero)
- A method **getTotal()** that returns the total amount of money, in dollars, collected by this meter.

a) Write a class **MeterTest** with a main method that tests the **ParkingMeter** class above under the following scenario:

- *A new parking meter is constructed with 15 minutes per quarter and a maximum of 60 minutes*
- *Three quarters are inserted*
- *20 minutes passes*
- *(*)*
- *Four quarters are inserted*
- *90 minutes passes*
- *(*)*

At the points marked (*), your test should check the values returned by the accessor methods. Print out the *actual* value obtained from calling the method, and then print out the *correct* value that you expect. No other output is required.

b) Write a complete implementation of the **ParkingMeter** class described above.

4. (14 pts) Write an interactive main method that takes a measurement in meters and converts it to either centimeters (1/100 meters) or kilometers (1000 meters). The method starts by prompting the user to enter the number of meters. Then the method asks whether the user wants to convert to centimeters (cm) or kilometers (km). If the user gives an invalid response to the question the program ends with an error message. Finally, the method outputs the converted measurement. The output must match the examples exactly, except for the formatting of numbers (e.g., significant digits) which can be ignored. A sample interaction is shown below, with user's responses in ***bold***.

Example 1:
How many meters? **150.25** Convert to cm or km? ***km*** The measurement is: 0.015025 km
Example 2:
How many meters? **150.25** Convert to cm or km? ***cm*** The measurement is: 15025 cm
Example 3:
How many meters? **10** Convert to cm or km? ***miles*** The requested conversion is not possible.

Write your code after the TODO comment.

```
import java.util.Scanner;
public class TimeConverter {
    public static void main(String[] args) {
        // TODO: your code here
```

5. (14 pts) Write a **static method** to guess Steve's birthday using an instance of java.util.Random! All we know for sure is that it is in September (which has 30 days) and that he was born sometime between 1900 and 1999. Your method should **return a String** of the form "September X, Y" where X is your randomly generated day from 1 through 30, and Y is your randomly generated year from 1900 through 1999. The method has no input.

Write your code after the TODO comment.

```
import java.util.Random;
public class SomeClass {
    public static String guessBirthday() {
        // TODO: your code here
```

6. (14 pts) Each row of the following table contains two code samples. In the third column write "same" or "different" to indicate if the code samples achieve the same result (in terms of setting the value of resultBool or resultNum) in all cases. Assume the following variables, with unknown values:

```
int x, y, z;
boolean condition;
```

Code Sample 1	Code Sample 2	"same" or "different"

`resultBool = x > y && is;`	`resultBool = false;`
	`if (is) {`
	`  if (x > y) {`
	`  resultBool = true;`
	`  }`
	`}`	

`resultBool = (x == y || x > z)`
`  && x != z;`
	`resultBool = false;`
	`if (x == y) {`
	`  if (x > z) {`
	`  resultBool = true;`
	`  }`
	`}`
	`if (x != z) {`
	`  resultBool = true;`
	`}`	

`resultBool = !(x < y)`
`  || (x < y && x != z);`
	`resultBool = false;`
	`if (x >= y) {`
	`  resultBool = true;`
	`}`
	`if (x < y) {`
	`  if (x != z) {`
	`  resultBool = true;`
	`  }`
	`}`	

`resultBool =`
`  (x > y + 2 || x < y - 2)`
`  && x != z;`
	`resultBool = false;`
	`if (x != z) {`
	`  if (x > y + 2) {`
	`  resultBool = true;`
	`  } else if (x < y - 2) {`
	`  resultBool = true;`
	`  }`
	`}`	

`resultNum = 0;`
`if (x > 0) {`
`  resultNum = 5;`
`}`
`if (y > 0) {`
`  resultNum = 7;`
`}`
	`resultNum= 0;`
	`if (x > 0) {`
	`  resultNum = 5;`
	`} else if (y > 0) {`
	`  resultNum = 7;`
	`}`	

`resultBool = x > y;`	`resultBool = !(x < y);`	

`resultNum = 0;`
`if (x == y) {`
`  resultNum++;`
`}`
`if (x == z) {`
`  resultNum++;`
`}`
	`resultNum = 0;`
	`if (x == y && x != y) {`
	`  resultNum++;`
	`} else if (x != y && x == y) {`
	`  resultNum++;`
	`} else if (x == y && y == x) {`
	`  resultNum+=2;`
	`}`	

`resultNum = 0;`
`if (x < 10) {`
`  resultNum++;`
`}`
`if (x > 20) {`
`  resultNum++;`
`}`
	`resultNum = 0;`
	`if (x < 10) {`
	`  resultNum++;`
	`} else if (x > 20) {`
	`  resultNum++;`
	`}`	

Rewrite the method **foo()** so that it does exactly the same thing but does not use any conditional statements (no "**if**" or "**if/else**" statements, ternary expressions, **while**-loops, **for**-loops, etc.). (Partial credit may be given if you do it with just one conditional statement.)

`public boolean foo(int x, int y) {`
`  boolean result = false;`
`  if (x > 0) {`
`  if (y / x == 2) {`
`  result = true;`
`  } else if (y / x == 3) {`
`  result = true;`
`  }`
`  }`
`  return result;`
 	
`}`
	`public boolean foo(int x, int y) {`
	
	`}`

Excerpts from the Java API documentation

Excerpt from documentation for java.lang.String

Method Summary	
`char charAt(int index)`	Returns the character at the given index.
`boolean contains(String s)`	Returns true if the given string is a substring of this string.
`boolean equals(String other)`	Returns true if this string is the same as the given string
`int indexOf(char ch)`	Returns the index of the first occurrence of the given character in this string. Returns -1 if the this string does not contain the given character.
`int indexOf(String s)`	Returns the index of the first occurrence of the given string s as a substring of this string. Returns -1 if s is not a substring.
`int lastIndexOf(char ch)`	Returns the index of the last occurrence of the given character in this string. Returns -1 if the this string does not contain the given character.
`int lastIndexOf(String s)`	Returns the index of the last occurrence of the given string s as a substring of this string. Returns -1 if s is not a substring.
`int length()`	Returns the length of this string.
`String substring(int beginIndex)`	Returns a substring of this string starting at beginIndex.
`String substring(int beginIndex, int endIndex)`	Returns a substring of this string consisting of the characters from beginIndex through endIndex `- 1`.
`String toUpperCase()`	Returns a copy of this string with all alphabetic characters converted to upper case.
`String toLowerCase()`	Returns a copy of this string with all alphabetic characters converted to lower case.
`String trim()`	Returns a copy of this string with leading and trailing whitespace removed.

Excerpt from documentation for java.lang.Math

Method Summary	
`static double abs(double x)`	Returns the absolute value of the given number.
`static int abs(int x)`	Returns the absolute value of the given integer.
`static double max(double x, double y)`	Returns the larger of the two given numbers.
`static int max(int x, int y)`	Returns the larger of the two given integers
`static double min(double x, double y)`	Returns the smaller of the two given numbers.
`static int min(int x, int y)`	Returns the smaller of the two given integers
`static double pow(double x, double y)`	Returns x to the power y.

`static long round(double x)`	Returns the given number rounded to the nearest whole number value (can be cast to int)
`static double sqrt(double x)`	Returns the square root of the given number.

Excerpt from documentation for java.lang.Integer

Method Summary	
`static int parseInt(String s)`	Converts the given string to an int value, if possible.

Excerpt from documentation for java.lang.Double

Method Summary	
`static double parseDouble(String s)`	Converts the given string to a double value, if possible.

Excerpt from documentation for java.util.Scanner

Constructor Summary	
`Scanner(InputStream source)`	Constructs a new scanner that provides values scanned from a given input stream, such as System.in.
`Scanner(String source)`	Constructs a new scanner that uses the given string as its input stream.
Method Summary	
`boolean hasNext()`	Returns true if this scanner has another token in its input.
`boolean hasNextDouble()`	Returns true if the next token in this scanner's input can be parsed as a floating-point value.
`boolean hasNextInt()`	Returns true if the next token in this scanner's input can be parsed as an int value.
`boolean hasNextLine()`	Returns true if this scanner's input contains a newline character.
`String next()`	Returns the next token in this scanner's input.
`double nextDouble()`	Returns the next token in this scanner's input, converted to a floating-point value, if possible.
`int nextInt()`	Returns the next token in this scanner's input, converted to an int value, if possible.
`String nextLine()`	Returns all input up to the next newline character.
`void useDelimiter(String pattern)`	Sets this scanner's delimiter to the given string (default delimiter is whitespace).

Excerpt from documentation for java.util.Random

Constructor Summary	
`Random()`	Constructs a new random number generator.
Method Summary	
`int nextInt(int max)`	Returns a pseudorandom, uniformly distributed value between 0 (inclusive) and **max** (exclusive), drawn from this generator's sequence.