Computer Sciences 302
Midterm Exam 2, 20%

Thursday 11/15, 2012

Print last (family) name: ____________________, first: ____________

Signature: ____________________ CS login: ____________

Circle Your Lecture	Lec 1	Lec 2	Lec 3	Lec 4	Lec 5	Lec 6	Lec 8
	Deb Deppeler	Deb Deppeler	Stephen Brown	Melissa Tress	Seth Pollen	Ben Miller	Shree Hardikar

Before you Begin:

(1) Turn in your UW student ID.
(2) **On this examination booklet:**
- Print and sign your name above.
- Write your CS login **and circle your lecture above.**

(3) Check that there is a total of 16 pages in this exam.
(4) You may not use any notes, books, calculators (or any other electronic devices), or neighbors on this exam. Turn off and put away your cell phone, pager, pda, etc. now.
(5) The exam is intended to take about 100 minutes, but **we will give you 2 hours to complete the exam**.
(6) We can't provide hints but if you need an exam question clarified or feel that there is an error, please bring this to our attention. If needed, **corrections will be written on the board**.
(7) Note: The amount of space for answer choices does not imply you must have that much code. There is double the answer space we think that most students will need to solve each question.

When you've Finished:

(8) Turn in this examination booklet and make sure we return your ID.

Taking the Exam

Write your answers to the questions in this examination booklet. Write neatly and format your Java code by indenting and spacing for readability. Comments are not required, *but if something isn't specified, do something reasonable and state your assumptions*.

Note a REFERENCE is provided at the END OF THE EXAM, which you should review when the exam begins.

Parts	Number of Questions	Topic	Possible Points	Score
I	1	Coding a Complete Program	20	
II	2	Using Pre-built Classes	20	
III	1	Coding an Instantiable Class	20	
IV	2	Using Arrays and ArrayLists	12	
		Total	72	

Part I Coding a Complete Program *[1 question, 20 points]*

[/ 20 points] Use good programming practices to write a complete Java program that allows the program user to determine the charges on an invoice for a local dairy delivery service. An invoice is a document that itemizes a customer's charges (money owed) and credits (money back). Your program must do the following.
You may abbreviate `System.out.print()` statements with `S.o.p()`.

- **Input two integers** corresponding to the number of half-gallon bottles (1/2 gallon per bottle) of milk ordered, and the number of empty bottles returned for credit by the customer. The minimum order is three one half-gallon bottles of milk. There is no minimum number of empty bottles that must be returned for an order. You may assume the user input is the correct type, but if a user input value is outside of the valid range of values, display an appropriate error message and allow a new value to be entered.
- Compute and display each of the following (in this order):
  - **Milk Subtotal**: Milk price is $2.75 for each half-gallon bottle. Subtotal is the cost of all milk ordered.
  - **Bottle Deposit Charge**: $1.50 for each half-gallon bottle ordered.
  - **Bottle Return Refund**: $1.50 for each empty bottle returned.
  - **Delivery Charge**: For milk orders less than $20 (not including the bottle deposit or return) there is a delivery charge. If the milk order is less than $10 the delivery charge is $2.50, otherwise the delivery charge is $1.50.
  - **Total Bill:** the milk subtotal plus the bottle deposit charge minus the bottle return refund amount plus the delivery charge.
  - **Cost Per Gallon ($/gal)**: cost of "milk and delivery" (dollars) / quantity (gallons) on this order. ***Caution: this result will rarely be an integer and it is computed as the (cost of milk plus the delivery charge) divided by the number of gallons (not bottles) of milk.***
- End.

Sample Run of a program that answers this question correctly: User responses in bold type.

```
Number of half gallons? 2
Error! Please enter an integer >= 3: 3
Number of bottles returned? -1
Error! Please enter an integer >= 0: 4
Milk Subtotal:           $8.25
Bottle Deposit Charge: +  4.5
Bottle Return Refund:  -  6.0
Delivery Charge:       +  2.5
Total Bill:            = $9.25
Cost Per Gallon:         $7.166666666666667/gal
```

Start your answer Part I here.

If necessary, continue your answer Part I here.

Part II Using Pre-built Classes *[2 questions, 20 total points]*

1.) [/ 10 points total] Full credit is given for answers that make best use of the methods available for these Java pre-built classes. Consult the reference page as needed.

Write a code fragment that replaces the string `"Wis. "` or `"Wisc. "` with the characters `"WI "` in a string named `city`. For example, if `city` started out referencing the string "Windsor, Wis. 53598" , after your code fragment, `city` would reference "Windsor, WI 53598".

Your code fragment may assume that either `"Wis. "` or `"Wisc. "` (but not both), may occur zero or one time in the `city` string and therefore your fragment must only find and replace the first instance of either string. Correct solutions will not assume other information about the strings and city state zip errors must not crash your code. Note letter case and the space character in the match strings; this implies that instances of "wis." "WIS" "WISC" "Wisc " "wisc." should not be matched and replaced.

```
String city = //assume this references a city state and zip code
```

2.) [/ 10 points] Consider the following classes and these public fields and methods which have been defined previously:

`Config` class stores several constants used by a game, including:

```
final int GAME_HEIGHT  //the height in pixels of the game space
final int GAME_WIDTH   //the width in pixels of the game space
```

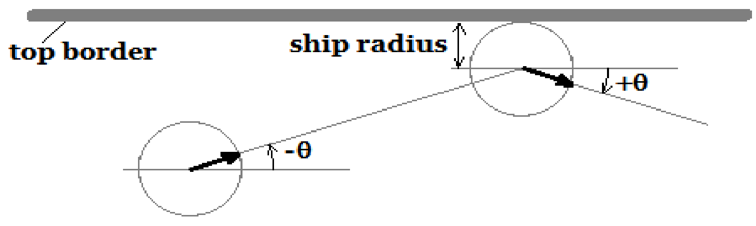
The `Position` class stores the position of a `Ship` as the distance from the origin along each axis

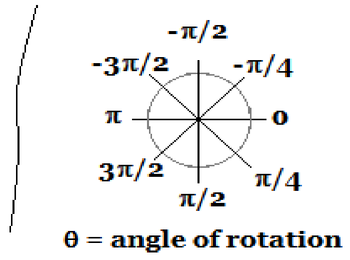
```
int getX()             //the distance along x-axis from left most pos 0
int getY()             //the distance along y-axis from top most pos 0
```

The `Ship` class represents a game player's ship in the game program:

```
boolean destroyed      //true if ship has been destroyed
double getRadius()     //returns the radius of the ship
Position getPosition() //returns the current position of the ship
double getRotation()   //returns the angle of rotation from horizontal
void updatePosition(double t) //moves the ship based on current position,
                              //speed, rotation and elapsed time (t)
void setRotation(double r) //sets the angle of rotation from horizontal
void destroy() //destroys this ship (not visible, not collidable, etc.)
```

Complete the TODO portions of the `update()` method (see next page) of the `Ship` class so that it: (a) the ship is destroyed if any part of it has moved past the left or right border, and (b) the ship "bounces" if it next to (touches) the top or bottom wall. A ship bounces by changing its angle of rotation when it is next to the top or bottom border. See figure.





The figure shows how the angle of rotation would change for a ship moving from left to right with an initial angle of rotation -θ changing to its new "bounced" angle of rotation +θ. A correct implementation will correctly change the ship's state when it touches (is next to) the top or bottom border or it is past the left or right border. No other collisions must be considered in this problem. A negative angle indicates angle in radians when direction is measured counter-clockwise from horizontal pointing right. A positive angle of rotation indicates an angle in radians when measured clockwise from horizontal to the right.

```
/**
 * Updates ship position and rotation based on ship having
 * moved in current direction and speed for t milliseconds.
 * @param t The number of milliseconds since last move.
 */
public void update(double t) {

   if ( ! destroyed ) {
      updatePosition(t);  // moves ship in current direction for t ms

      //TODO bounce if any part of ship is next to top or bottom border



      //TODO destroy if any part of ship has passed left or right border



   } // end if

} // end update() method
```

Part III Coding an Instantiable Class *[1 question, 20 points]*

[/ 20 points] Implement an instantiable class, named `ProblemGen`, used to represent a binary arithmetic random problem generator to be used in a quiz program. Binary arithmetic problems have two operands, an operator, and an answer, e.g. 2 + 3 = 5, has 2 as its first operand, 3 as its second operand, ADD as its operator, and 5 as its answer; 15 - -4 = 19, has 15 as its first operand, -4 as its second operand, SUB as its operator, and 19 as its answer.

Each problem generator keeps: a random number generator, a minimum value for operand values, a maximum operand value, an arithmetic operation code indicating what type of binary problem this problem generator generates (addition, subtraction, multiplication, or division), a current question (binary expression), and a current answer (double). When a `ProblemGen` object is created, it also randomly generates the first question's operands and operator and computes and sets its answer field. A `ProblemGen` object can be used to generate new problems with the same operator, by randomly generating different operands (within the established range), and recomputing the correct answer. Use good object-oriented programming practices. *If an invalid operator code integer is specified when a ProblemGen instance is created, use '?' for the displayed operator character and 0.0 for the answer.*

// SUGGESTED PRIVATE MEMBERS (additional private members may be necessary)

- A random number generator
- The smallest (minimum) possible value for either operand. (int)
- The largest (maximum) possible value for either operand. (int)
- The arithmetic operator used for this problem generator. (int)
- The current arithmetic question. (String)
- The correct answer for the current problem. (double)

// REQUIRED PUBLIC MEMBERS (no additional public members may be added)

- Operator code constants (int): `ADD`, `SUB`, `MUL`, `DIV`
- `ProblemGen(int min, int max, int operator)` creates and initializes a problem generator's fields. Parameter's must be: the minimum operand value (int), the maximum operand value (int), and the arithmetic operator (int) to use. *For proper encapsulation, ensure that all instance fields are properly initialized when an instance is constructed.*
- `toString()` returns a `String` representation of the current question and answer. The string must include the binary expression and its answer, e.g. `"15 - -4 = 19"`.
- `getQuestion()` returns just the question expression as a string of characters, e.g. "`15 - -4`"
- `isCorrect(double ans)` - returns `true` if `ans` is within +/-0.001 of the computed correct answer. Using a range like this avoids most rounding issues for division problems.
- `nextQuestion()` generates another question and answer using the same operator and operand ranges, but it does not return any value. The `getQuestion()` method would have to be called to see the new question string.

Example of a `ProblemGen` object being used:

```
ProblemGen p = new ProblemGen( 1, 20, ProblemGen.SUB );
System.out.print("What is " + p.getQuestion() + "? "); // displays the question
double ans = stdin.nextDouble();   // assumes that a Scanner named stdin is available
if ( p.isCorrect(ans) )            // checks the user's answer against correct answer
    System.out.println("Correct! " + p ); // if correct, display results.
p.nextQuestion(); // Then, create a new question and answer using same Problem generator object
```

Use this page to answer Part III.

If necessary, continue your answer to Part III.

Part IV Using Arrays and ArrayLists *[2 questions, 12 total points]*

1.) **[/ 6 points]** Assume `team` is a properly constructed `ArrayList` of `Player` objects that has been filled with at least five players. The class `Player` contains an accessor method, `getPoints()` that returns the number of *points* scored by that player as in `int`. **Write a code fragment** that displays the player or players who scored the most points. Assume that the `Player` class has a properly defined `toString()` method. *Hint: determine highest points scored, then display players who have scored that many points.*

2.) [/ 6 points] In program 2, you completed an Epidemic simulator program that simulated the spread of a contagious disease. Recall, that the integer values in the array were used to indicate the health status of individuals in the world array.

Write a complete method, named `countIndividuals`, that is passed a 2D array of **int** values representing the world of individuals, and it returns an array of **int** values. The first element in the returned array is the number of **`UNINFECTED`** individuals, the second element in the array is the number of **`RECOVERED`** individuals, and the third value is the number of remaining individuals (not **`EMPTY`**). *You may assume the 2D array passed to the method is properly constructed, and that each row has the same number of columns, and that Config.UNINFECTED, Config.RECOVERED, and Config.EMPTY are public static constants with the correct integer used to indicate each of those states.*

Exam Reference Page

Methods from the `java.util.ArrayList` class (*REMEMBER 0-based indexing is used):

```
Note the E's below are replaced with the particular ArrayList's element type.
void add(E item)        // Adds the specified item to the end of this list.
void add(int index, E item)// Adds the specified item by inserting it into this
                           // list at the specified index.
boolean contains(E item)// Returns true iff the specified item is in this list,
                        // otherwise false.
E get(int index)        // Returns the item at the specified index in this list.
int indexOf(E item)     // Returns the index of the specified item if it is in
                        // this list, or -1 if the item is not in this list.
E remove(int index)     // Removes and returns the item from the specified index.
boolean remove(E item)  // Returns true iff the specified item was removed from
                        // this list, otherwise false.
int size()              // Returns the number of used elements in this list.
```

Methods from the `java.lang.Math` class:

```
double abs(double n)    //Returns the absolute value of n.
double ceil(double n)   //Returns the value of n rounded up to the nearest
                        //whole number.
```

Methods from the `java.util.Random` class:

```
Random()                //Creates a new random number generator.
Random(int s)           //Creates a new random number generator seeded with s.
int nextInt(int n)      //Returns the next pseudo-random integer value
                        //between 0 (inclusive) and n (exclusive).
```

Methods from the `java.util.Scanner` class:

```
Scanner(System.in)      // Creates a Scanner object that reads from the keyboard.
boolean hasNext()       // Returns true if there's another token of input.
boolean hasNextDouble() // Returns true if the next input is a double value.
boolean hasNextInt()    // Returns true if the next input is an int value.
boolean hasNextLine()   // Returns true if there's another line of input.
String next()           // Returns the next token of input.
double nextDouble()     // Scans the next token of the input as a double.
int    nextInt()        // Scans the next token of the input as an int.
String nextLine()       // Returns the next line of input as a String.
```

Methods from the `java.lang.String` class (*REMEMBER 0-based indexing is used):

```
String(String str)      // Creates a String object given another string.
char charAt(int index)  // Returns the char value at the specified index.
boolean contains(String s)// Returns true iff string s is in this string,
                        // otherwise false.
boolean equals(String s)// Returns true iff the contents of this string
                        // is the same as that of string s.
boolean equalsIgnoreCase(String s)// Returns true iff the contents of the
                                  // this string is the same as that of the
                                  // string s, ignoring differences in case.
int indexOf(String s)   // Returns the index within this string of the first
                        // character of the first occurrence of the specified
                        // string s.
int length()            // Returns the length of this string.
String toLowerCase()    // Returns a new string that is the lowercase
                        // version of this string.
String substring(int beginIndex)
                        // Returns a new string that is a substring of this string
                        // starting at beginIndex to the end of the this string.
String substring(int beginIndx, int endIndx)
                        // Returns a new string that is a substring of this string
                        // starting at beginIndx up to but not including endIndx
```

Print last name: ____, first: _________________

Signature: ____ CS login: _________________

This sheet must be returned with the rest of your exam.
If you remove this sheet, fill in the information above and tuck it between the pages of your exam when you turn it in.