|  |  |
| --- | --- |
|  | **University of Maryland College Park** |
| **Dept of Computer Science** |
| **CMSC132 Summer 2013** |
| **Midterm I Key** |

First Name (PRINT): **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Last Name (PRINT): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

University Directory ID (e.g., testudoJr) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

I pledge on my honor that I have not given or received any unauthorized assistance on this examination.

Your signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Instructions**

* This exam is a closed-book and closed-notes exam.
* Total point value is 200 pts.
* The exam is a 50 minutes exam.
* Please use a pencil to complete the exam.
* **WRITE NEATLY**.

**Grader Use Only**

|  |  |  |  |
| --- | --- | --- | --- |
| #1 | Java Language Features | (50) |  |
| #2 | Arrays | (50) |  |
| #3 | Class Implementation | (100) |  |
| **Total** | Total | (200) |  |

**Problem 1 (50 pts) Java Language Features**

1. (6 pts) Provide an example that illustrates procedural abstraction.

**Answer:** One example: a function that sorts data, but we don’t indicate how to sort it.

1. (4 pts) Mention one Java language feature that allow us to implement encapsulation.

**Answer:** The **private** access specifier.

1. (6 pts) You would like to generate a message when an object is garbage collected. Would you be able to generate that message using the finalize() method? Briefly explain.

**Answer:** Yes, as finalize() is called when an object is garbage collected.

1. (6 pts) When should you use an enumerated type vs. a class? Briefly explain.

**Answer:** When you have a fixed number of instances (e.g., phases of the moon).

1. (4 pts) For our first project (ClearCellGame) which component (of the Model-View-Controller) was provided?

**Answer:** Both the View and Controller.

1. (10 pts) After the provided code, write down and/or cross out errors (if any) or unnecessary code (if any), present in the **equals** method associated with the Telephone class. Two Telephone objects are considered equal if they have the same string value.

**public class Telephone {**

**private String number;**

**public boolean equals(Telephone obj) {**

**if (this == null || obj != this) {**

**return true;**

**} else if (obj != null && !(obj instanceof Telephone)) {**

**return false;**

**}**

**return number.equals(((Telephone)(obj)).number);**

**}**

**}**

**Answer:**

1. Parameter needs to be Object instead of Telephone.
2. **this == null** is unnecessary (this can never be null).
3. **obj != null** is unnecessary (instanceof takes care of this case).
4. (4 pts) What is difference between making a variable **final static** vs. just **final**? In other words, what is the difference between the following two declarations? Briefly explain.

**final static int x;**

**final int x;**

**Answer:** Using static defines a single constant for the whole class, whereas non-static defines a constant per object.

1. (10 pts) In the main method, define an anonymous class instance that overrides the fly() method so that the message “supersonic” is printed (instead of “slow flying plane”).

**public class Plane {**

**public void fly() {**

**System.*out*.println("slow flying plane");**

**}**

**}**

**public class Driver {**

**public static void main(String[] args) {**

**Plane supersonic =**

**supersonic.fly(); // this should print “supersonic”**

**}**

**}**

**Answer:**

Plane supersonic = new Plane() {

public void fly() {

System.out.println("supersonic");

}

};

**Problem 2 (50 pts) Arrays**

Implement a method called **prefixProduct** that returns a new array where each array element at index k corresponds to the product of elements of the array **src** starting at index 0 and including the element at index k. For example, for array {2, 3, 5} the method will return array {2, 6, 30}. For an array of size 0 or a null parameter, the method will throw the exception **IllegalArgumentException** with the message “Invalid argument”. For this problem the **src** array may not be modified, and you may not use ArrayList.

**One Possible Answer:**

**public static int[] prefixProduct(int[] src) {**

**if (src == null || src.length == 0)**

**throw new IllegalArgumentException("Invalid argument");**

**int[] result = new int[src.length];**

**for (int currentMax = 0; currentMax < src.length; currentMax++) {**

**int prod = 1;**

**for (int i = 0; i <= currentMax; i++)**

**prod \*= src[i];**

**result[currentMax] = prod;**

**}**

**return result;**

**}**

**Problem 3 (100 pts) Class Implementation**

For this problem you need to implement two classes: **Game** (an abstract class) and **FlipCellGame** (a class that extends Game). Notice that your implementation must be efficient, and it should have the minimum number of methods and instance variables possible.

1. **Game** class –Abstract class with the following specification:
   1. **board** 🡪 Two dimensional array of integers that can be accessed by classes extending this class.
   2. **Constructor** 🡪 Takes two parameters: maximum number of rows, and maximum number of columns. The constructor will create a two-dimensional array of integers, and will initialize all the entries of the array to 1.
   3. **processThisCell** 🡪 This method makes the class abstract. The method takes as parameters two integers (rowIndex, colIndex) that represent the cell that will be processed.
2. **FlipCellGame** class – This class has the **Game** class as superclass. In addition, it implements the Comparable interface. The class has the following specification:
   1. **Constructor** 🡪 Takes two parameters: maximum number of rows and maximum number of columns. The constructor will create a two-dimensional array of integers, and will initialize all the entries of the array to 1.
   2. **processThisCell** 🡪 This method will flip a cell. That is, if the cell has the value 1 it will assign 0, and vice versa. You can assume the arguments provided are within the board.
   3. The **Comparable** interface implementation will allow us to sort FlipCellGame objects in such a way that objects with boards having a small number of rows will appear first.

The sample driver (and output) below can help you verify the functionality of some of the methods described above. **Notice the sample output relies on a toString() method that you do not need to implement.**  **Feel free to ignore this driver if you know what to implement.**

**Sample Driver**

**FlipCellGame fg = new FlipCellGame(4, 7);**

**System.*out*.println(fg);**

**fg.processThisCell(2, 3);**

**System.*out*.println(fg);**

**fg.processThisCell(2, 3);**

**System.*out*.println(fg);**

**FlipCellGame fg2 = new FlipCellGame(2, 7);**

**System.*out*.println(fg.compareTo(fg2) > 0);**

**Sample Driver Output**

**Game Board**

**1111111**

**1111111**

**1111111**

**1111111**

**Game Board**

**1111111**

**1111111**

**1110111**

**1111111**

**Game Board**

**1111111**

**1111111**

**1111111**

**1111111**

**true**

**Answer:**

1. **(38 pts) Game** class

**public abstract class Game {**

**protected int[][] board;**

**public Game(int maxRows, int maxCols) {**

**board = new int[maxRows][maxCols];**

**for (int row = 0; row < board.length; row++) {**

**for (int col = 0; col < board[row].length; col++) {**

**board[row][col] = 1;**

**}**

**}**

**}**

**public abstract void processThisCell(int rowIndex, int colIndex);**

**}**

1. **(62 pts) FlipCellGame** class

**public class FlipCellGame extends Game implements Comparable<FlipCellGame> {**

**public FlipCellGame(int maxRows, int maxCols) {**

**super(maxRows, maxCols);**

**}**

**public void processThisCell(int rowIndex, int colIndex) {**

**if (board[rowIndex][colIndex] == 0) {**

**board[rowIndex][colIndex] = 1;**

**} else {**

**board[rowIndex][colIndex] = 0;**

**}**

**}**

**public int compareTo(FlipCellGame fg) {**

**return board.length - fg.board.length;**

**}**

**}**