German University in Cairo
Media Engineering and Technology Faculty
Prof. Dr. Slim Abdennadher

June 14, 2014

CSEN202: Introduction to Computer Programming Spring Semester 2014

Final Exam

Bar Code

Instructions: Read carefully before proceeding.

1) Duration of the exam: 3 hours (180 minutes).
2) No books or other aids are permitted for this test.
3) This exam booklet contains 17 pages, including this one. Three extra sheets of scratch paper are attached and have to be kept attached. **Note that if one or more pages are missing, you will lose their points. Thus, you must check that your exam booklet is complete**.
4) Write your solutions in the space provided. If you need more space, write on the back of the sheet containing the problem or on the four extra sheets and make an arrow indicating that. **Scratch sheets will not be graded unless an arrow on the problem page indicates that the solution extends to the scratch sheets**.
5) When you are told that time is up, stop working on the test.

Good Luck!

Don't write anything below ;-)

Exercise	1	2	3	4	5	6	$\sum$
Possible Marks	10	12	12	12	12	27	85
Final Marks							

Exercise 1 (10 Marks)

Write a method `has271` that given an array of integers returns true if the array contains a `2, 7, 1` pattern; that is, a value, followed by the value plus 5, followed by the value minus 1.

Here are some examples of evaluating the method on representative input values:

```
has271({1, 2, 7, 1}) -> true
has271({1, 2, 8, 1}) -> false
has271({2, 7, 1})    -> true
has271({4, 9, 3})    -> true

public static boolean has271(int[] nums) {
```

Solution:

```java
public class Has271 {
    public static boolean has271(int[] nums) {
        for (int i = 0; i < nums.length - 2; i++) {
            if ((nums[i + 1] == nums[i] + 5) && (nums[i + 2] == nums[i] - 1)) {
                return true;
            }
        }
        return false;
    }
}
```

Exercise 2 (12 Marks)

Write a recursive method `reverseRec` that given an array of integers reverses the elements of the given array.

Note that you are not allowed to use any additional arrays.

Running the following `main` method:

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

    int[] a = {1,2,3,4};
    int[] b = {5,2};
    reverseRec(a);
    reverseRec(b);
}
```

array `a` will be equal to `{4,3,2,1}` and array `b` will be equal to `{2,5}`.

```
public static int[] reverseRec(int[] a) {
```

Solution:

```
public static int[] reverseRec(int[] a) {
    return reverseHelper(a,0,a.length -1);
}

public static int[] reverseHelper (int[] a, int start , int end) {
    if (start >= end) {
        return a;
    }
    else {
        // this code will swap two elements
        int temp = a[start];
        a[start] = a[end];
        a[end] = temp;
    }
    return reverseHelper (a, start+1, end-1);
}
```

Exercise 3 (12 Marks)

Write a method that, given a two-dimensional array of numbers, prints a checker box selection of the elements of the array.

Hint: To print a checker box selection, keep alternating *between* printing elements in odd positions of each row *and* printing elements in even positions of each row (once the odd, once the even)

Example. For the following array

```
1    3    17   3    12
51   11    4   6     2
1    3     3   7     0
3    1     6   2     7
4    2     3   6     9
```

your program should print:

```
     3         3
51         4         2
     3         7
3          6         7
     2         6
```

or (alternatively and likewise correctly) print:

```
1         17        12
   11          6
1          3         0
     1         2
4          3         9
```

Solution:

```java
public static void checkerBox(int[][]x)
{
    for(int i =0; i< x.length;i++)
    {
        for(int j=0; j<x[i].length; j++)
        {
            if(j%2==i%2) // or if(j%2!=i%2) to get the alternative solution
                System.out.print(x[i][j]);
            else
                System.out.print("␣");
        }
        System.out.println();
    }
}
```

Exercise 4 (12 Marks)

The aim of this question is to create a class `Interval` which represents an interval of the real numbers. For example, the closed interval `[2.5, 5.0]` would represent all the numbers between 2.5 and 5.0 inclusive.

Implement a class `Interval` with the following:

a) `public Interval(double left, double right)`

   Class constructor. The values of `left` and `right` are the two endpoints of the interval.

b) `public boolean doesContain(double x)`

   Returns true if and only if `x` lies between left and right, and the interval is not empty (i.e. `left < right`).

c) `public boolean isEmpty()`

   Returns true if and only if `left` is larger than `right`.

d) `public boolean intersects(Interval b)`

   Returns true if and only if the Interval `b` intersects with this interval. Note that empty intervals cannot intersect with other intervals.

e) `public String toString()`

   Returns a string summarising the Interval; for example

   ```
   "Interval: [2.5, 5.0]"
   ```

   or

   ```
   "Interval: (EMPTY)"
   ```

Solution of Exercise 4

Solution:

```
public class Interval {

    private double left, right;

    public Interval(double left, double right) {
        this.left = left;
        this.right = right;
    }

    public boolean doesContain(double item) {
        if (isEmpty()) {
            return false;
        }
        return (left < item && right > item);
    }

    public boolean isEmpty() {
        return left > right;
    }

    public boolean intersects(Interval other) {
        if (isEmpty() || other.isEmpty()) {
            return false;
        }

        if (left > other.right) {
            return false;
        }
        if (right < other.left) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        if (isEmpty()) {
            return "Interval:␣(EMPTY)";
        }
        return "Interval:␣[" + left + ",␣" + right + "]";
    }

}
```

Exercise 5 Objects and references (12 Marks)
Lego bricks

The following classes manage a pile of colored Lego^{TM}-bricks. Please consider the class `Color` and the class `Brick`.

Further consider the tester class `Bricklayer`, which instantiates some colors and bricks.

```
public class Color {
String name;

public Color (String name) {
    this.name = name;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}
}


public class Brick {
static int nextSerial = 0;
int serial;
Color color;

public Brick () {
    this.serial = nextSerial++;
}

public Brick (Color color) {
    this();
    this.color = color;
}

public Color getColor() {
    return color;
}

public void setColor(Color color) {
    this.color = color;
}

public void display() {
    System.out.println("Brick " + serial + " (" + color.getName() + ")");
}
}
```

```
public class Bricklayer {

public static void main(String[] args) {
  Color red = new Color ("Red");
  Brick redBrick = new Brick (red);

  Color blue = new Color ("Blue");
  Brick blueBrick = new Brick (blue);

  Brick yellowBrick = new Brick (new Color ("Yellow"));

  Color green = red;
  green.setName("Green");
  Brick greenBrick = new Brick (green);

  Brick orangeBrick = greenBrick;
  orangeBrick.getColor().setName("Orange");

  yellowBrick.setColor(blue);

  Brick blackBrick = new Brick ();

}

public static boolean compare1(Brick a, Brick b) {
    return a == b;
}

public static boolean compare2(Brick a, Brick b) {
  return a.getColor() == b.getColor();
}

public static boolean compare3(Brick a, Brick b) {
  return a.getColor().getName() == b.getColor().getName();
}
}
```

a) Give the exact output once we include the following lines to the `main` method:

```
redBrick.display();
blueBrick.display();
yellowBrick.display();
greenBrick.display();
orangeBrick.display();
```

Solution:

```
Brick 0 (Orange)
Brick 1 (Blue)
Brick 2 (Blue)
Brick 3 (Orange)
Brick 3 (Orange)
```

b) Give the exact output once we include the following lines to the `main` method:

```
 System.out.println("The blocks are "
        + (compare1(orangeBrick, greenBrick) ? "equal" : "distinct"));
 System.out.println("The blocks are "
    + (compare2(orangeBrick, greenBrick) ? "equal" : "distinct"));
 System.out.println("The blocks are "
    + (compare3(orangeBrick, greenBrick) ? "equal" : "distinct"));
```

Solution:

```
The blocks are equal
The blocks are equal
The blocks are equal
```

c) Give the exact output once we include the following lines to the `main` method:

```
System.out.println("The blocks are "
    + (compare1(blueBrick, yellowBrick) ? "equal" : "distinct"));
System.out.println("The blocks are "
    + (compare2(blueBrick, yellowBrick) ? "equal" : "distinct"));
System.out.println("The blocks are "
    + (compare3(blueBrick, yellowBrick) ? "equal" : "distinct"));
```

Solution:

```
The blocks are distinct
The blocks are equal
The blocks are equal
```

d) Give the output once we include the following line to the `main` method:

```
blackBrick.display();
```

Solution:

`NullPointerException` since the object `blackBrick` was created using the default constructor that initializes `blackBrick.color` with `null`.

Exercise 6 (27 + 4 (Bonus) Marks)

The goal of this exercise is to design and implement a simple inventory control system for a small video rental store.

Here is a summary of the design specification:

- Store a library of videos identified by title.
- Allow a video to indicate whether it is currently rented out.
- Allow users to rate a video and display the average rating for the video.
- Get a list of all the videos that are currently checked out of the store.

You will define two classes: a `Video` class to model a video and a class `VideoStore` class to model the store.

Assume that an object of class `Video` has the following attributes:

- a title;
- a boolean flag to say whether it is checked out or not.

In addition, you will need to store information about the ratings which the video has received.

In the class `Video` you are not allowed to use arrays.

The class `Video` should implement the following:

a) `public Video(String title)`

   A constructor that takes the title of the video as a parameter.

b) `public String getTitle()`

   Return the video's title.

c) `public boolean addRating(int rating)`

   Add a rating for the video. If rating is between 1 and 5 inclusive, then update the ratings for this video, keeping track of how many ratings for it have been received, and return true. Otherwise, print out an error message and return false.

d) `public double getAverageRating()`

   Return the average rating for this video. Although ratings are always integers, the average should be a double. Return zero if no ratings have been added.

e) `public boolean checkOut()`

   If the video is already checked out, warn the user and return false. Otherwise change the status of the video to checked out, and return true.

f) `public boolean returnToStore()`

   If the video is not checked out, warn the user and return false. Otherwise change the status of the video to not checked out, and return true.

g) `public boolean isCheckedOut()`

   Return the checked-out status of the video

h) `public String toString()`

   Return a String of the form

   ```
   Video[title="<title>", checkedOut=<status>]
   ```

Implemetation of the `Video` **class (I)**

Solution:

```
public class Video {

    private final String title;
    private boolean checkedOut = false;
    private int ratingSum = 0;
    private int ratingCount = 0;

    public Video(String title) {
        this.title = title;
    }

    public String getTitle() {
    return title;
    }

    public boolean addRating(int rating) {
        if (rating < 1 || rating > 5) {
            System.out.println(rating + "␣should␣be␣between␣1␣and␣5.");
            return false;
        }
        ratingSum += rating;
        ratingCount++;
        return true;
    }

    public double getAverageRating() {
        if (ratingCount == 0) {
            return 0;
        }
        return ((double) ratingSum) / ratingCount;
    }

    public boolean checkOut() {
        if (checkedOut) {
            System.out.println(this + "␣is␣already␣checked␣out.");
        return false;
        }
        checkedOut = true;
        return true;
    }

    public boolean returnToStore() {
        if (!checkedOut) {
            System.out.println(this + "␣is␣not␣checked␣out.");
            return false;
        }
        checkedOut = false;
        return true;
    }

    public boolean isCheckedOut() {
        return checkedOut;
    }
```

Implemetation of the `Video` **class (II)**

Solution:

```
    @Override
    public String toString() {
        return "Video[title=\"" + title + "\", checkedOut=" + checkedOut + "]";
    }
}
```

The `VideoStore` class should contain an array containing all the videos in the store's inventory, and has the following:

a) `public VideoStore(int capacity)`

   A constructor that creates an array of videos with a limited capacity.

b) `public Video getVideo(String title)`

   Return the video whose title is title. If there is no video in the inventory with that title, print out an error message and return `null`.

c) `public boolean addVideo(String title)`

   Add a video by title to the inventory. If there is already a video with that title in the store's inventory or the array is full with videos, print out an error message and return `false`. Otherwise, add a new video with that title and return `true`.

d) `public boolean checkOutVideo(String title)`

   Check out a video by title. If there is a video with that title not already checked out, change its status to checked out and return true. Otherwise, print out an appropriate error message and return false.

e) `public Video[] getCheckedOut()`

   Return an array of type Video[] consisting of all videos in the store which have been checked out. This method is a bonus.

Implemetation of the `VideoStore` **class**

Solution:

```
public class VideoStore {
    private Video[] videos;

    public VideoStore(int capacity){
        videos = new Video[capacity];
    }
    public Video getVideo(String title) {
        for (int i=0; i<videos.length;i++) {
            if (videos[i] != null && videos[i].getTitle().equals(title))
                return videos[i];
        }
        System.out.println("Sorry, cannot find the requested video in the catalogue"
        return null;
    }
    public boolean addVideo(String title) {
        for (int i=0; i<videos.length;i++) {
            if (videos[i] != null && videos[i].getTitle().equals(title)) {
                System.out.println(title + " is already in stock.");
                return false;
            }
        }
        for (int i=0; i<videos.length;i++) {
            if (videos[i]==null) {
                videos[i] = new Video(title);
                return true;
            }
        }
                System.out.println("There's no space in the store");
        return false;
    }
    public boolean checkOutVideo(String title) {
        Video video = getVideo(title);
        if (video != null)
            return video.checkOut();
        return false;
    }
    public Video[] getCheckedOut() {
        int count = 0;
        int i;
        for (i=0; i<videos.length;i++) {
            if (videos[i]!=null && videos[i].isCheckedOut())
                count++;
        }
        Video[] checkedOut = new Video[count];
        for (i=videos.length-1; i>=0;i--) {
            if (videos[i]!=null && videos[i].isCheckedOut()) {
                checkedOut[count-1] = videos[i];
                count--;
            }
        }
        return checkedOut;
    }
}
```

Scratch paper

Scratch paper

Scratch paper