Nick Troccoli Practice Midterm

CS 106A July 18, 2017

Solutions to Practice Midterm

Based on handouts by Mehran Sahami, Eric Roberts and Patrick Young

Problem 1: Karel the Robot (20 points)

**/\* File: InnerBorderKarel.java \*/**

**import stanford.karel.\*;**

**public class InnerBorderKarel extends SuperKarel {**

**public void run() {**

**moveUpRow();**

**for(int i = 0; i < 4; i++) {**

**handleBorder();**

**nextPosition();**

**}**

**}**

**/\* Assumes Karel starts one avenue before the first beeper to**

**\* be placed in this line of the border. Places beepers until**

**\* Karel reaches a wall, but does not place a beeper on the last**

**\* corner (where Karel is facing the wall).**

**\*/**

**private void handleBorder() {**

**move();**

**while (frontIsClear()) {**

**// We check for any existing beepers, so we don't put**

**// two beepers on any of the "corners" of the border**

**if (noBeepersPresent()) {**

**putBeeper();**

**}**

**move();**

**}**

**}**

**// Moves Karel up one row while keeping the same orientation**

**private void moveUpRow() {**

**turnLeft();**

**move();**

**turnRight();**

**}**

**/\* Assumes Karel is facing a wall at the end of line of placed**

**\* beepers and repositions Karel to be facing in direction of next**

**\* line in the border of beepers that needs to be placed.**

**\*/**

**private void nextPosition() {**

**turnRight();**

**move();**

**turnRight();**

**move();**

**turnRight();**

**}**

**}**

Problem 2: Java expressions, statements, and methods (20 points)

|  |  |  |
| --- | --- | --- |
| **(2a)** | **5.0 / 4 - 4 / 5** | 1.25 |
|  | **7 < 9 - 5 && 3 % 0 == 3** | **false** |
|  | **"B" + 8 + 4** | **"B84"** |

**(2b)**

The 1st number is: 78

The 2nd number is: 73

Problem 3: Console Programs (25 points)

/\*

\* File: SecondLargest.java

\* ------------------------

\* This program finds the largest and second largest number

\* in a list entered by the user.

\*/

import acm.program.\*;

public class SecondLargest extends ConsoleProgram {

// Defines the sentinel used to signal the end of the input

private static final int SENTINEL = 0;

public void run() {

println("This program finds the two largest integers in a");

println("list. Enter values, one per line, using a "

+ SENTINEL + " to");

println("signal the end of the list.");

int largest = -1;

int secondLargest = -1;

int input = readInt(" ? ");

while (input != SENTINEL) {

if (input > largest) {

secondLargest = largest;

largest = input;

} else if (input > secondLargest) {

secondLargest = input;

}

input = readInt(" ? ");

}

println("The largest value is " + largest);

println("The second largest is " + secondLargest);

}

}

Problem 4: Graphics Programs (20 points)

|  |  |  |
| --- | --- | --- |
|  | /\*  \* File: Frogger.java  \* ------------------------  \* This program solves the Frogger problem from the practice midterm,  \* where the frog jumps vertically based on the position of mouse  \* clicks.  \*/  import acm.graphics.\*;  import acm.program.\*;  import java.awt.\*;  import java.awt.event.\*;  public class Frogger extends GraphicsProgram {  private GImage frog;  public void run() {  // Just for testing purposes; try changing window size here  setCanvasSize(300, 220);    frog = new GImage("res/frog.gif");  double fx = (getWidth() - frog.getWidth()) / 2;  double fy = getHeight() - frog.getHeight();  add(frog, fx, fy);  }  public void mouseClicked(MouseEvent event) {  double mouseY = event.getY();  double frogTop = frog.getY();  double frogHeight = frog.getHeight();  double frogBottom = frogTop + frogHeight;  if (mouseY < frogTop && frogTop >= frogHeight) {  frog.move(0, -frogHeight);  } else if (mouseY > frogBottom &&  frogBottom + frogHeight <= getHeight()) {  frog.move(0, frogHeight);  }  }  } |  |

Problem 5: Strings, Characters and Files (35 points)

(2a) removeDuplicates

|  |  |  |
| --- | --- | --- |
|  | private String removeDuplicates(String str) {  String result = "";  for (int i = 0; i < str.length(); i++) {  char ch = str.charAt(i);  if (i == 0 || ch != str.charAt(i - 1)) {  result += ch;  }  }  return result;  } |  |

**(2b) removeDuplicatesFromFile**

|  |  |  |
| --- | --- | --- |
|  | private void removeDuplicatesFromFile(String filename) {  try {  Scanner input = new Scanner(new File(filename));  // We need to read line by line to preserve line breaks  while (input.hasNextLine()) {  String line = input.nextLine();  Scanner tokens = new Scanner(line);  while (tokens.hasNext()) {  String word = tokens.next();  print(removeDuplicates(word) + " ");  }  println();  }  input.close();  } catch (IOException fnfe) {  println("file could not be read");  }  } |  |