

Practice Midterm Solutions

Problem 1: Karel is Lost!

(24 Points)

Here are a few possible solutions:

```
public void run() {  
    /* Get to the nearest wall. */  
    while (frontIsClear() && noBeepersPresent()) {  
        move();  
    }  
  
    /* Until we find a door, run around the room  
    * hugging the wall. This implicitly stops  
    * running if we found the beeper initially.  
    */  
    while (noBeepersPresent()) {  
        turnLeft();  
        while (frontIsClear() && rightIsBlocked()) {  
            move();  
        }  
        /* If we found the door, walk outside. This  
        * case handles both that we've walked out  
        * already and where the door is to our  
        * side.  
        */  
        if (noBeepersPresent() && rightIsClear()) {  
            turnRight();  
            move();  
        }  
    }  
}
```

```
public void run() {  
    /* Get to the nearest wall. */  
    while (frontIsClear() && noBeepersPresent()) {  
        move();  
    }  
    /* Orient so that we're now facing down the  
    * wall we hit.  
    */  
    turnLeft();  
  
    /* Execute the "hug the wall" strategy. */  
    while (noBeepersPresent()) {  
        if (rightIsClear()) {  
            turnRight();  
            move();  
        } else if (frontIsClear()) {  
            move();  
        } else {  
            turnLeft();  
        }  
    }  
}
```

This problem is tricky because there are many cases to check. Karel might start facing right out the door, or the door might be at the end of a wall. The most common mistakes we saw were neglecting to cover these cases and minor mistakes involving getting left and right mixed up.

Problem Two: Jumbled Java hiJinks**(20 Points Total)****(i) Expression Tracing****(6 Points)**`1 - 2 - 3 - 4``-8``(13 / 7) / (7 / 13)``Error: Divide by 0``1 == 2 || 2 / 0 == 3``Error: Divide by 0``"1" + (1 + 1) + 1 + 1 + "1"``"12111"`

The first expression evaluates to -8 because it is parsed as

`((1 - 2) - 3) - 4`

The second expression causes a division by zero error. Since 7 and 13 are **ints**, 7 / 13 uses integer division, rounding down to 0. Dividing 13 / 7 by 0 then causes the error.

The third expression also causes a division by zero error. Java first evaluates `1 == 2`, and since it is **false** it does not short-circuit and then evaluates the right-hand side, causing a division by 0 error.

The final expression is evaluated from the left to the right. This concatenates the string `"1"` with the value of `(1 + 1)`, then concatenates another `1`, then concatenates another `1`, then finally concatenates the string `"1"` to the end.

(ii) Program Tracing**(14 Points)**

The correct answer is

```

robert = 7
abraham = 56
maryTodd = 61
maryTodd = 61
maryTodd = 12
abraham = 7
abraham = 12
maryTodd = 56

```

The most common error was updating **abraham** in **run**, even though **abraham** was not reassigned.

Problem Three: Nim**(32 Points)**

Here is one possible solution:

```
public class Nim extends ConsoleProgram {
    /* Minimum number of stones in each pile at the start of the game. */
    private static final int MIN_STONES = 1;
    private static final int MAX_STONES = 20;
    public void run() {
        RandomGenerator rgen = RandomGenerator.getInstance();
        int pile1 = rgen.nextInt(MIN_STONES, MAX_STONES);
        int pile2 = rgen.nextInt(MIN_STONES, MAX_STONES);

        while (true) {
            for (int player = 1; player <= 2; player++) {
                println("Player " + player + "'s turn.");
                println("Pile 1:" + pile1 + " stone(s)Pile 2: " + pile2 + " stone(s).");
                if (pile1 == 0 && pile2 == 0) {
                    println("Player " + player + " wins!");
                    return;
                }

                int pile = choosePile(pile1, pile2);
                if (pile == 1) {
                    pile1 -= chooseStones(pile1);
                } else {
                    pile2 -= chooseStones(pile2);
                }
            }
        }
    }
    private int choosePile(int pile1, int pile2) {
        while (true) {
            int pile = readInt("Choose a pile: ");
            if ((pile == 1 && pile1 != 0) || (pile == 2 && pile2 != 0)) return pile;
            println("Please choose a nonempty pile.");
        }
    }
    private int chooseStones(int stonesInPile) {
        while (true) {
            int number = readInt("Remove how many stones? ");
            if (number >= 1 && number <= stonesInPile) return number;
            println("Please enter a valid number.");
        }
    }
}
```

The most common mistakes we encountered had to do with error-checking. Many solutions included a **while** loop like this one to check for a valid pile:

```
while (pile != 1 || pile != 2) {  
    /* Reprompt */  
}
```

This goes into an infinite loop, since any value of **pile** will be not equal to 1 or not equal 2.

Other common mistakes included running the game in a loop like this:

```
while (pile1 != 0 && pile2 != 0) {  
    /* Play the game! */  
}
```

This loop will terminate as soon as either pile becomes empty, instead of looping while at least one pile is nonempty.

Problem Four: Picture Panel Programs**(22 Points)**

```

import acm.program.*;
import acm.graphics.*;
import java.awt.*;
import java.awt.event.*;

public class PicturePanelProgram extends GraphicsProgram {
    /* Name of the file containing the image to hide. */
    private static final String IMAGE_FILENAME = "puppy.jpg";
    private GImage background = new GImage(IMAGE_NAME);

    public void run() {
        add(background);
        addFrontLayer();
        addMouseListeners();
    }

    private void addFrontLayer() {
        double width = getWidth() / 4.0;
        double height = getHeight() / 4.0;

        for (int x = 0; x < 4; x++) {
            for (int y = 0; y < 4; y++) {
                GRect panel = new GRect(x * width, y * height, width, height);
                panel.setFilled(true);
                panel.setColor(Color.WHITE);
                panel.setFill(Color.GRAY);
                add(panel);
            }
        }
    }

    public void mouseClicked(MouseEvent e) {
        GObject hit = getElementAt(e.getX(), e.getY());
        if (hit != background) {
            remove(hit);
        }
    }
}

```

Common mistakes included accidentally making it possible to remove the background by clicking on it, and incorrectly sizing or positioning the blocks.

Problem Five: Damaged DNA Diagnoses**(22 Points)**

```
private int costOfDNAErrorsIn(String one, String two) {
    int totalCost = 0;
    for (int i = 0; i < one.length(); i++) {
        totalCost += costOf(one.charAt(i), two.charAt(i));
    }
    return totalCost;
}

private int costOf(char a, char b) {
    if (a == '-' || b == '-') return 2;
    if (b != matchOf(a)) return 1;
    return 0;
}

private char matchOf(char a) {
    if (a == 'A') return 'T';
    if (a == 'T') return 'A';
    if (a == 'C') return 'G';
    return 'C';
}
```

Common mistakes included iterating over the string incorrectly and not correctly tracking the cost of the mismatches properly.