Practice Midterm Examination

Midterm exam: Tuesday, July 17, 7:00 P.M.–8:30 P.M., Location: Hewlett 200

This handout is intended to give you practice solving problems that are comparable in format and difficulty to those which will appear on the midterm examination this Wednesday. A solution set to this practice examination will be distributed on Tuesday.

Time and place of the exam
The midterm exam is scheduled at the time and place above. If there are any special concerns please send an e-mail message to ojimenez@stanford.edu.

Coverage
The exam covers the material presented in class through around Monday, July 11, which means that you are responsible for the Karel material plus the chapters of the Art and Science of Java outlined in the course calendar that we asked you to read, Except for most of Chapter 8. Because you have not had experience writing programs using the String class, you won’t be asked to generate any strings. The only operator we expect for you guys to know is how to use the + operator when working with strings.

Note: To conserve trees, I have cut back on the answer space for the practice midterm. The actual exam will have much more room for your answers and for any scratch work. We also won’t be expecting you to be writing the public void run or any of the constants that we have mentioned unless we mention that explicitly in the handout.

Please remember that the midterm is open-book.

General instructions
Answer each of the five questions included in the exam. Write all of your answers directly on the examination paper, including any work that you wish to be considered for partial credit.

Each question is marked with the number of points assigned to that problem. The total number of points is 60. We intend for the number of points to be roughly comparable to the number of minutes you should spend on that problem. This leaves you with an additional hour to check your work or recover from false starts.

In all questions, you may include methods or definitions that have been developed in the course, either by writing the import line for the appropriate package or by giving the name of the method and the handout or chapter number in which that definition appears.

Unless otherwise indicated as part of the instructions for a specific problem, comments will not be required on the exam. Uncommented code that gets the job done will be
sufficient for full credit on the problem. On the other hand, comments may help you to
get partial credit if they help us determine what you were trying to do.

The examination is open-book, and you may make use of any texts, handouts, or course
notes. You may not, however, use a computer of any kind.

**Problem 1: Karel the Robot (10 points)**

After Hurricane Katrina, many communities along the Gulf Coast were reduced to rubble.
To help rebuild those shattered cities and towns, the government has established a new
agency named Katrina Automated RELief (or KAREL) whose mission is to dispatch
house-building robots to repair the damaged area. Your job is to program those robots.

Each robot begins at the west end of a street that might look like this:

![Graphical representation of a street with houses and beacons]

Each beeper in the figure represents a pile of debris where a house once stood. Karel’s job
is to walk along the street and build a new house in the place marked by each beeper.
Those houses, moreover, need to be raised on stilts to avoid damage from the next storm.
Each house, in fact, should look exactly like this:

![House with stilts]

The new house should be centered at the point at which the bit of debris was left, which
means that the first house in the diagram above will be constructed with its left edge
along 2nd Avenue.

At the end of the run, Karel should be at the east end of the street having created a set of
houses that look like this for the initial conditions shown:

![Graphical representation of houses at the end of the street]

In solving this problem, you can count on the following facts about the world:

- Karel starts off facing east at the corner of 1st Street and 1st Avenue with an infinite
  number beepers in its beeper bag.
• The beepers indicating the positions at which houses should be built will be spaced so that there is room to build the houses without overlapping or hitting walls. Note that this condition implies that there cannot be a beeper in the first or last intersection of the roadway. Note also that Karel needs to pick up these beepers given that the final configuration leaves no beepers in their original positions.

• Karel’s must end up facing east at the southeast corner of the world. Moreover, Karel should not run into a wall if it builds a house that extends into that final corner.
Problem 2: Simple Java expressions, statements, and methods (10 points)

(2a) Compute the value of each of the following Java expressions. If an error occurs during any of these evaluations, write “Error” on that line and explain briefly why the error occurs.

\[
\begin{align*}
5.0 &/ 4 - 4 / 5 \\
7 &< 9 - 5 \&\& 3 \% 0 == 3 \\
"B" &+ 8 + 4
\end{align*}
\]

(2b) What output is printed by the following program:

```java
/*
 * File: Problem2c.java
 * ----------------------
 * This program doesn't do anything useful and exists only to test
 * your understanding of method calls and parameter passing.
 * *
 */
import acm.program.*;
public class Problem2c extends ConsoleProgram {
    public void run() {
        enigma(11);
    }
    private void enigma(int x) {
        int y = 0;
        for (int i = 1; i < 2 * x; i *= 2) {
            y = puzzle(y, mystery(x, i));
        }
        println("enigma: x = " + x + ", y = " + y);
    }
    private int puzzle(int x, int y) {
        x = 10 * x + y;
        println("puzzle: x = " + x + ", y = " + y);
        return x;
    }
    private int mystery(int x, int y) {
        return x / y % 2;
    }
}
```
Problem 3 – Simple Java (15 points) - A random day on the Caltrain!

You're bored one day and fed up with testing your breakout; what to do? Spend a day visiting random cities in the bay area on the Caltrain!

For this problem, you will simulate riding the Caltrain to `NUM_STOPS` different stops, starting here in Palo Alto. Assume you have a normal 6-sided die and regular coin, and at each stop you will
  a) flip the coin to see if you are going north or south
  b) roll the die to see how many stops you are going (between 1 and 6).
  c) repeat this process for a total of `NUM_STOPS` times.

An example run of your program might look like this:

Start at station #10
Go north 4 stations to station #14
Go north 4 stations to station #18
Go south 6 stations to station #12
Go south 2 stations to station #10
Go north 2 stations to station #12
End at station #12

Other details:

- Assume there are `NUM_STATIONS` in the system, and that they are numbered from 0 (Tamien) to 26 (San Francisco).
- At each stop, you should print out which direction you went (north or south), how many stops you travelled (1 to 6), and the number of the station that you stopped at. You should also print out the starting and ending stations.
- it's ok if you end up visiting the same station more than once.
• Assume that you will make \texttt{NUM\_STOPS} stops over the course of the day.
  Your program should work for any value of \texttt{NUM\_STOPS}.

Now here’s an additional wrinkle. If you are near either end of the train route, and your
die roll / coin flip would run you off the end of the track, you should ignore them and try
again until you get a valid station to go to.

For example, if you are at station 3, and your roll tells you to go \texttt{south 5} stations, you
should flip the coin and roll the die again. Keep doing so until you get a valid station to
visit, but be sure to print an error message each time. You program needs to run until you
visit \texttt{NUM\_STOPS valid} stations.

Here’s an example of this:

\begin{verbatim}
Start at station #10
Go south 6 stations to station #4
Go south 1 stations to station #3
Go south 5 stations -- Out of bounds. Roll again.
Go south 6 stations -- Out of bounds. Roll again.
Go north 2 stations to station #5
Go south 3 stations to station #2
Go south 3 stations -- Out of bounds. Roll again.
Go north 3 stations to station #5
End at station #5
\end{verbatim}

\textbf{Problem 4 – (Graphics Program) – FAIL stamp (10 points)}

For this problem you are going to make a \texttt{GraphicsProgram} that draws a simple stamp
that responds to the user's mouse movements. Here's how it will work:
you will have one \texttt{GImage} for the stamp (use "stamp.jpg"). This should be \texttt{centered}
around the user's mouse. As the user moves the mouse around the screen, the image
should also move.

• When the user presses the mouse, it will simulate adding a stamped image to the
screen. Do that by the following:
  1. the stamp \texttt{GImage} should change to the "stampPressed.jpg" image when
     the mouse is pressed.
  2. when the mouse is released, you should go back to the "stamp.jpg"
     image.
  3. Also when the mouse is released you should add a new image to the screen
     ("fail.jpg"), centered where the mouse was released. The new image
     represents the impression the stamp made on the canvas.

Here’s an example showing the stamp moving around the screen:
Here we show the user pressing the mouse, releasing the mouse, and then moving the mouse to the right. Notice the stamped image is added to the screen, and it just sits there.

Before pressing the mouse
mouse moves away

while the mouse is pressed

the mouse is released

(the ‘fail’ image is added, but it’s under the stamp)

Here’s a screenshot after a few stampings:
The images are:
“stamp.jpg”  “stampPressed.jpg”  “fail.jpg”

Also note: You should only use instance variables that are appropriate. We will mark off for poor use of instance variables.

Write your solution below.

```java
public class FailStamp extends GraphicsProgram {

    private static final String STAMP = “stamp.jpg”;
    private static final String PRESSED = “stampPressed.jpg”;
    private static final String FAIL = “fail.jpg”;

    // Add instance variables here

    public void run() {

        // Write your code here
    }
}
```