Problem 1: Karel the Robot  (10 points)

Seeing as how it’s getting on toward Valentine’s Day, Karel has decided to make valentines (represented by beepers, of course) for every student in the elementary school class that is using Karel to learn about programming. Karel does not remember exactly how many desks there are in each horizontal row but does remember that there are precisely four rows of desks and that the classroom looks something like the one shown in the following diagram:

Karel may count on the following facts:

1. Karel starts at 1st Avenue and 1st Street, facing east, with an infinite number of beepers in its bag.
2. There are four rows of student desks, positioned as shown in the diagram, just to the north of 2nd, 4th, 6th, and 8th Streets.
3. Karel does not know how many desks there are in each row (which may not all be the same), or how many blank spaces there are between the desks, or how many spaces exist between the desks at the ends of each row and the walls of the classroom. What Karel does know is that each of the desks is exactly one unit wide and that there are
no desks right up against the wall, which means that there is always a corridor along 1\(^{st}\) Avenue for Karel to walk from one row to the next.

When Karel is done, all of the desks in the room should have a Valentine, as shown in the following diagram:

```

Karel may end up facing any direction on any convenient square.

Write the Karel program necessary to deliver valentines to all of the desks in the room. As always, your task is to get them to the desks, and you need not try to be particularly efficient about it.

In your solution, you may use any Karel method defined in the course handouts just by giving its name. For example, you may use `turnRight` or `moveToWall` without writing down the complete definition.

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.

\[
5.0 / 4 - 4 / 5
\]

\[
7 < 9 - 5 && 3 \% 0 == 3
\]

\[
"E" - "A"
\]
(2b) Assume that the method `mystery` has been defined as given below:

```java
private int mystery(int n) {
    while (n >= 10) {
        int k = 0;
        while (n > 0) {
            k += n % 10;
            n /= 10;
        }
        n = k;
    }
    return n;
}
```

What is the value of

`mystery(1729)`

(2c) 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 parameters and string methods.
 */
import acm.program.*;
public class Problem2c extends ConsoleProgram {
    public void run() {
        String s1 = "Heart";
        String s2 = valentine("candy", s1);
        println("s1 = " + s1);
        println("s2 = " + s2);
    }
    private String valentine(String s1, String s2) {
        int num = (s1.substring(1, 2)).length();
        s1 = s2.substring(num);
        s2 = cupid(s1, s2.charAt(0));
        return s2;
    }
    private String cupid(String s1, char ch) {
        return (s1 + Character.toLowerCase(ch));
    }
}
```
Problem 3: Simple Java programs (15 points)
As you undoubtedly learned in school, the Pythagorean Theorem holds that the length of the hypotenuse \( z \) of a right triangle with sides \( x \) and \( y \) is given by the following formula:

\[ x^2 + y^2 = z^2 \]

As it turns out, there are an infinite number of triangles in which all three of these edge lengths are integers, including the following examples:

\[
\begin{array}{c}
3^2 + 4^2 = 5^2 \\
5^2 + 12^2 = 13^2
\end{array}
\]

Because of this connection to the Pythagorean Theorem, any set of integers \( x, y, \) and \( z \) that meets this condition is called a Pythagorean triple.

Write a Java program that prints out all Pythagorean triples in which both \( x \) and \( y \) are less than or equal to a named constant \( \text{LIMIT} \) and \( x \) is less than \( y \). For example, if \( \text{LIMIT} \) is 25, your program should generate the following sample run:

In writing this problem, you should keep the following points in mind:

- You should not worry at all about efficiency. Trying every possible pairing of \( x \) and \( y \) and seeing whether it works is perfectly acceptable.
- The \texttt{Math.sqrt} method returns a \texttt{double}, which is only an approximation. It is possible, for example, that \texttt{Math.sqrt(25)} returns a \texttt{double} that is ever so slightly different from 5, which means that it might be come out as 4.9999999999999999 or 5.0000000000000001. The important point is that it might not be equal to an integer. Thus, to check whether an integer is a perfect square, you have to make the final test in the \texttt{int} domain where computation is exact. In case it comes in handy, there is a method \texttt{GMath.round(x)} that rounds a \texttt{double} to the nearest \texttt{int}. 

Problem 4: Using the graphics and random number libraries (15 points)

You never see a frog so modest and straightforward as he was, for all he was so gifted. And when it come to fair and square jumping on a dead level, he could get over more ground at one straddle than any animal of his breed you ever see.

—Mark Twain, “The Notorious Jumping Frog of Calaveras County,” 1865

As you may recall, the winner in the aesthetic division of this year’s Karel Contest played the game of Frogger, in which the object is to guide a frog across a screen filled with moving cars, floating logs, and alligators. Such a game is beyond the scope of an exam problem, but it is relatively straightforward to write the code that (1) puts an image of the frog on the screen and (2) gets the frog to jump when the user clicks the mouse.

Your first task in this problem is to place the frog at the bottom of the graphics window, as shown on the right. The frog itself is the easy part because all you need to do is create a `GImage` object with the appropriate picture, as follows:

```java
GImage frog = new GImage("frog.gif");
```

The harder part is getting the image in the appropriate place in the bottom of the window. In Frogger, the frog image cannot be just anywhere on the screen but must instead occupy a position in an imaginary grid such as the one shown on the right. The size of the grid is controlled by three named constants, which have the following values for this grid:

```java
public static final int SQSIZE = 75;
public static final int NCOLS = 7;
public static final int NROWS = 3;
```

The `SQSIZE` constant indicates that each of the squares in the grid is 75 pixels in each dimension and the other two parameters give the width and height of the grid in terms of the number of squares. Remember that the squares shown in the most recent diagram do not actually exist but simply define the legal positions for the frog. In the initial position, the frog must be in the center square along the bottom row. You may assume `NCOLS` is odd so that there is a center square, and you may also assume that `APPLICATION_WIDTH` and `APPLICATION_HEIGHT` have been set so the `NCOLS x NROWS` squares fill the window.

The second part of the problem is getting the frog to jump when the user clicks the mouse. The goal is to get the frog to jump one square in the direction that moves it closest to the mouse. For example, if you click the mouse at the location shown in the diagram at the right, the frog should move `SQSIZE` pixels upward so that it occupies the center square in the grid. If the user then clicked the mouse at the left edge of the screen, the frog should jump `SQSIZE` pixels to the left. The frog, however, should never jump outside the window.

The following restatement of the rule may clarify the intended behavior more explicit. The frog should jump one square position in the direction—up, down, left, or right—that corresponds most closely to the direction from the center of the frog to the mouse position. Thus, in the diagram, the frog should move up rather than right because the distance to the mouse is larger in the y direction than it is in the x direction. If, however, the new position would lie outside the `NCOLS x NROWS` grid, the frog should stay where it is.
Problem 5: Strings and characters (10 points)

*How do I love thee? Let me count the ways.*
—Elizabeth Barrett Browning, *Sonnet 43*, 1850

Computers are, of course, very good at counting things. Write a method

```java
private int countLove(String str)
```

that takes a string as its argument and returns the number of times the word *love* appears in that string, ignoring differences in case, but making sure that *love* is not just part of a longer word like *clover*, *glove*, *pullover*, or *slovenly*. For example, if you were to call

```java
countLove("Love in the clover.")
```

your method should return 1. The word *Love* counts as a match because your method should ignore the fact that *Love* starts with an uppercase *L*. The word *clover* doesn’t match because the letters *love* are merely part of a larger word.

Although you have enough information to write the method already, some people find it helpful to see the method used in context. One possible use is illustrated by the following `run` method, which would count all the occurrences of *love* in an entire paragraph of text, ending with a blank line:

```java
public void run() {
    int count = 0;
    while (true) {
        String line = readLine();
        if (line.length() == 0) break;
        count += countLove(line);
    }
    println("Love occurs " + count + " times.");
}
```

This program might produce the following sample run, which uses Shakespeare’s Sonnet 116 as its input text:

```
Let me not to the marriage of true minds
Admit impediments, love is not love
Which alters when it alteration finds,
Or bends with the remover to remove.
O no, it is an ever-fixed mark
That looks on tempests and is never shaken;
It is the star to every wand'ring bark,
Whose worth's unknown, although his height be taken.
Love's not Time's fool, though rosy lips and cheeks
Within his bending sickle's compass come,
Love alters not with his brief hours and weeks,
But bears it out even to the edge of doom:
If this be error and upon me proved,
I never writ, nor no man ever loved.
```

Love occurs 4 times.

There are two matches in the second line, one in the ninth, and one in the eleventh. Note that *Love's* at the beginning of the ninth line counts because the apostrophe is not a letter, but that *loved* at the very end of the sonnet does not.