Problem 1: RectangleKarel (10 Points)

There are many solutions to this problem. The main challenges are not forgetting the first/last row/column, the first/last beeper in each row/column, and not crashing into a wall if Karel's initial position was flush up against a wall. Here's one possible solution:

```java
import stanford.karel.*;
public class RectangleKarel extends SuperKarel {
    public void run() {
        while (frontIsClear()) {
            makeColumn();
            move();
        }
        makeColumn();
    }

    /* Fills all corners below or at Karel’s position in the current column
     * with beepers, returning Karel back to the position Karel began in.
     * /
    private void makeColumn() {
        fillDown();
        comeBackUp();
    }

    /* Fills the corners below or at Karel’s position in the current column
     * with beepers. Karel begins facing East at some position and ends
     * facing South at the bottom of the column.
     * /
    private void fillDown() {
        turnRight();
        putBeeper();
        while (frontIsClear()) {
            move();
            putBeeper();
        }
    }

    /* Moves Karel from the bottom of a row, facing South, back up to the
     * position of the first beeper on the row, facing East.
     * /
    private void comeBackUp() {
        /* Ascend to the top of the world and descend down to the beepers. */
        turnAround();
        while (frontIsClear()) {
            move();
        }
        turnAround();
        while (noBeepersPresent()) {
            move();
        }
        turnLeft();
    }
}
```
Problem Two: Jumbled Java hiJinks (10 Points Total)

(i) The Magic Number (6 Points)

For each of these programs, determine whether there are any values of $x$ that can be entered so that the program will print *success* without causing any errors and without printing out anything else. If there are any values of $x$ that will work, give any one of them. If there are no values of $x$ that will work, write “no solution.” No justification is necessary.

```java
/* Program A */
int x = readInt();
if (x != 0 && x / 2 == 0) {
    println("success");
}
```

Answer for Program A: 1

```java
/* Program B */
int x = readInt();
if (x >= 10000) {
    while (x != 0) {
        if (x % 10 != 9) {
            println("failure");
        }
        x /= 10;
    }
    println("success");
}
```

Answer for Program B: 99999

```java
/* Program C */
int x = readInt();
if (x > 7 && x / 0 == x) {
    println("success");
}
```

Answer for Program C: no solution

```java
/* Program D */
int x = readInt();
if (x != 0 || x != 1) {
    println("failure");
}
println("success");
```

Answer for Program D: no solution

```java
/* Program E */
int x = readInt();
if (1 - 3 - 5 - x == -10) {
    println("success");
}
```

Answer for Program E: 3

```java
/* Program F */
int x = readInt();
if (x / 2 * 3 == 6) {
    println("success");
}
```

Answer for Program F: 4
(ii) Program Tracing

Determine the output of the following program and write it in the indicated box.

    import acm.program.*;
    public class FrozenJava extends ConsoleProgram {
        public void run() {
            int anna = 16;
            int elsa = 18;

            anna = arendelle(anna, elsa);
            println("anna = " + anna);
            println("elsa = " + elsa);
        }

        private int arendelle(int elsa, int anna) {
            String kristoff = "hans";

            weselton(kristoff);
            println("kristoff = " + kristoff);

            elsa = kristoff.length();
            return anna;
        }

        private void weselton(String olaf) {
            olaf += "el";
            println("olaf = " + olaf);
        }
    }

Write the output of this program in the box below:

    olaf = hansel
    kristoff = hans
    anna = 18
    elsa = 18
Problem Three: Slicing a Cake (10 Points)

import acm.program.*;
import acm.util.*;

public class SlicingACake extends ConsoleProgram {
    private static final int NUM_TRIALS = 10;

    public void run() {
        int numPeople = readInt("How many people? ");
        int totalHappy = 0;
        for (int i = 0; i < NUM_TRIALS; i++) {
            int happyNow = sliceACake(numPeople);
            println(" Round "+(i+1)+": "+happyNow);
            totalHappy += happyNow;
        }
        double result = (double)totalHappy / NUM_TRIALS;
        println("Average happy people: "+result);
    }

    private int sliceACake(int numPeople) {
        double cakeLeft = numPeople;
        int result = 0;

        RandomGenerator rgen = RandomGenerator.getInstance();
        while (cakeLeft >= 2) {
            cakeLeft -= rgen.nextDouble(1.0, 2.0);
            result++;
        }
        if (cakeLeft >= 1) result++;
        return result;
    }
}
Problem Four: Pebbling a Checkerboard  

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

public class PebblingACheckerboard extends GraphicsProgram {
    /* The size of a checker. */
    private static final double CHECKER_SIZE = 50;

    public void run() {
        addInitialCheckers();
        addMouseListeners();
    }

    /* Adds the initial three checkers to the world. */
    private void addInitialCheckers() {
        addChecker(0, getHeight() - CHECKER_SIZE);
        addChecker(0, getHeight() - 2 * CHECKER_SIZE);
        addChecker(CHECKER_SIZE, getHeight() - CHECKER_SIZE);
    }

    /* Adds a checker at the given position. */
    private void addChecker(double x, double y) {
        GOval checker = new GOval(x, y, CHECKER_SIZE, CHECKER_SIZE);
        checker.setFilled(true);
        add(checker);
    }

    /* Reacts to mouse clicks by adding checkers if appropriate. */
    public void mouseClicked(MouseEvent e) {
        GObject hit = getElementAt(e.getX(), e.getY());
        if (hit != null 
            && getElementAt(e.getX() + CHECKER_SIZE, e.getY()) == null 
            && getElementAt(e.getX(), e.getY() - CHECKER_SIZE) == null) {
            remove(hit);
            addChecker(hit.getX() + CHECKER_SIZE, hit.getY());
            addChecker(hit.getX(), hit.getY() - CHECKER_SIZE);
        }
    }
}
Problem Five: Damaged DNA Diagnoses  

Here’s one possible solution:

```java
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';
}
```