

## WES-CS GROUP MEETING #8

### Exercise 1: Designing and Implementing a Class

Below is an incomplete *CheckerBoard* class. When the main method executes, it should print the following output (representing a checkerboard with red and black squares):

```
R B R B R B R
B R B R B R B
R B R B R B R
B R B R B R B
R B R B R B R
B R B R B R B
R B R B R B R
```

Notice that the code uses a *CheckerBoardSquare* class. Your job is to write the *CheckerBoardSquare* class as well as the rest of the *CheckerBoard* class. To do this, you'll need to use the code given below to figure out what methods and data members each of the two classes needs to have, and what each method needs to do.

```
class CheckerBoard {

    // YOU MUST ADD CODE HERE

    public static void main(String[] args) {
        for (int i = 0; i < HEIGHT_OF_BOARD; i++) {
            for (int j = 0; j < WIDTH_OF_BOARD; j++) {
                CheckerBoardSquare square = new CheckerBoardSquare(i, j);
                square.printColor();
                System.out.print(' ');
            }
            endRow();
        }
    }
}
```

## Exercise 2: Loops

The six pictures shown below were printed by a simple Java program.

```
*
**
***
****
*****
*****
*****
*****

*
***
*****
*****
*****
*****
*****
*****

*****
*****
*****
*****
*****
*****
*****
*****

*****
*
*
*
*
*
*
*****

*****
**
* *
* *
* *
* *
* *
*****

*****
**
* *
* *
* *
* *
* *
*****
```

The first picture was printed by the program shown below.

```
class Loops {
    public static void printDesign1( int n ) {
        // print n rows of increasing length
        for (int row=0; row<n; row++) {
            for (int col=0; col<row+1; col++) {
                System.out.print('*');
            }
            System.out.println();
        }
        System.out.println();
    }

    public static void main(String[] args) {
        int n = 7;
        printDesign1( n );
    }
}
```

Trace the code to understand how it works. Then choose one or more of the other pictures, and add new methods that print the pictures you've chosen. (Make sure that your methods work for any value of  $n$  greater than 0.)

Now design your own picture (some pattern of stars in an  $n$ -by- $n$  box) and ask your neighbor to code it.

### Exercise 3: Mastermind (Logical Thinking *and* programming!)

Mastermind is a game for two players, the codemaker and the codebreaker. The game begins with the codemaker selecting a code: a sequence of four letters (c1, c2, c3, c4) chosen from a set of six letters (a, b, c, d, e, f) – repetitions are allowed. The codebreaker will then try to guess the code. Each guess is a sequence of four letters (g1, g2, g3, g4), and the codemaker responds with two numbers:

1. the number of exact matches (the number of letters in the guess that are the right letter in the right position))
2. the number of near matches (the number of letters in the guess that are the right letter, but in the wrong position)

The codebreaker has up to 8 chances to guess the correct code.

#### Part (a)

Play the game several times. Try to find a strategy for breaking the code. (You might be interested to know that you can always break the code in 5 guesses, and that there is a six-guess sequence that can break *all* codes!)

#### Part (b)

Now, we'll implement the codemaker player of the Mastermind game (so when you play the game, the program thinks of a code, and you try to guess it). An incomplete class is given on the last two pages; you need to implement the following methods to complete the class:

isWin(String guess):

return true if the codebreaker has guessed the code (i.e., *guess* is the same as the code)

exactMatches(String guess):

return the number of exact matches in *guess*

nearMatches(String guess):

return the number of near matches in *guess*

reportMatches(String guess):

print the number of exact matches and the number of near matches

A good strategy is to write and test the methods starting with the easiest ones and ending with the hardest ones. In this case, a good order is:

*isWin, reportMatches, exactMatches, nearMatches*

***Don't just write the methods, test each one!!***

The *nearMatches* method is the hardest because

- (a) An exact match doesn't count as a near match, so for example, if the code is "abcd" and the guess is "aeee", there is one exact match and no near matches.
- (b) Each near match in the guess can only match one letter in the code. For example, if the code is "abcd" and the guess is "eaaa", there are no exact matches, and only one near match.

You might find it useful in the *nearMatch* method to copy the code into a `StringBuffer`, where you can change the characters (to remove characters that have already been counted as exact or near matches). You can use the *setCharAt* method of the *stringBuffer* class to change one character:

```
public void setCharAt(int index, char ch)
```

```
The character at the specified index of this  
string buffer is set to ch. The index argument  
must be greater than or equal to 0, and less  
than the length of this string buffer.
```

```

import java.io.*;

/**
 * This class implements the Mastermind game. It will generate a random
 * code and the user has up to 8 chances to guess the correct code.
 */
public class Mastermind {

    // DATA MEMBERS //
    private String code;          // the code

    // CONSTRUCTOR //
    /**
     * randomly generate the code (a string with 4 letters
     * chosen from 'a' to 'f')
     */
    public Mastermind() {
        code = new String();
        for (int i = 0; i < 4; i++) {
            char ch = (char)(int)(Math.random() * 6 + 'a');
            code = code + ch;
        }
    }

    // PUBLIC METHODS
    /**
     * return the code
     */
    public String getCode() {
        return code;
    }

    /**
     * print the number of exact and near matches
     */
    public void reportMatches(String guess) {
        // NOT IMPLEMENTED YET
    }

    /**
     * return true if the guess is correct; otherwise, return false
     */
    public boolean isWin(String guess) {
        // NOT IMPLEMENTED YET
    }
}

```

```

// PRIVATE METHODS
/**
 * return the number of exact matches
 */
private int exactMatches(String guess) {
    // NOT IMPLEMENTED YET
}

/**
 * return the number of near matches
 */
private int nearMatches(String guess) {
    // NOT IMPLEMENTED YET
}

////////////////////////////////////
// MAIN METHOD: EXECUTION BEGINS HERE
////////////////////////////////////

public static void main( String [] args ) throws IOException {
    BufferedReader stdin =
        new BufferedReader( new InputStreamReader( System.in ) );
    Mastermind codemaker = new Mastermind();
    for(int i = 1; i <= 8; i++) {
        System.out.print("Guess: " + i + ": ");
        String guess = stdin.readLine();
        codemaker.reportMatches(guess);
        if( codemaker.isWin(guess) ) {
            System.out.println("You WIN!");
            System.exit(0); // Terminate if the user wins
        }
    }

    System.out.println("The correct code is " + codemaker.getCode());
}
}

```