

LCS-ES Group Meeting #4

Exercise 1: Using Objects and Methods

A delicious treat known as a *s'more* is constructed from the following ingredients:

- 1 graham cracker (broken in half)
- 2 chocolate rectangles
- 4 mini marshmallows

First, try making (and eating!) one s'more. Now let's design some Java programs where the objects are packages containing graham crackers, chocolate bars, and mini marshmallows. Assume that each package has one method: `numInPackage` that tells you how many items (graham crackers, chocolate bars, or mini marshmallows) are currently in the package.

Part (a). Describe in English how you would determine whether it's possible to make one s'more. How about two s'mores? n s'mores?

Part (b). Now assume that the package objects are called `crackerPkg`, `chocolatePkg`, and `marshmallowPkg`. Write Java code that determines whether it is possible to make one, two, or n s'mores and in each case either prints "yes" or "no".

Part (c). Describe in English how you would determine the *maximum* number of s'mores you can make using the ingredients in the three packages.

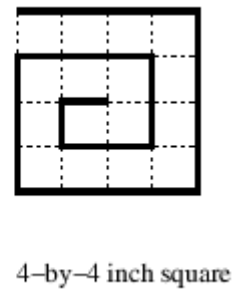
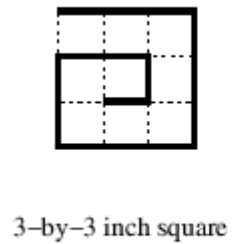
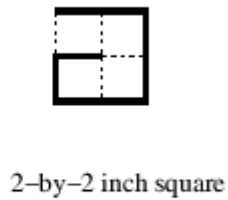
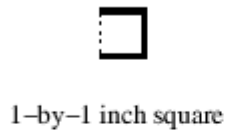
Part (d). Write Java code that prints the maximum number of s'mores that can be made.

Exercise 2: Writing a Java Method

Last week you did a long exercise involving the Artist class. One of the problems was to write code to draw a clockwise, square spiral in an n-by-n square. You may not have written actual code or you may have written code that works only for a particular n. This week we will write the general method. The method header is given below:

```
public void drawSpiral(int n)
```

And here are the pictures of some spirals to remind you what they look like.



Exercise 3: The Time class

For this exercise, we'll work on extending the Time class defined on the next page.

Part (a). First, try running the program as it is.

Part (b). Now, let's define some new methods for the Time class. Notice that the Time class stores its time in 24-hour format: it has two data members called *hour* and *minute*; *hour* is an integer between 0 and 23 that represents the hours part of the time, and *minute* is an integer between 0 and 59 that represents the minutes part of the time.

Our first new method will be `print12HourTime`, which is like the existing method `print24HourTime`, except that it prints the time in 12-hour format instead of 24-hour format. For example, the 24-hour time 15:20 would be printed as "3:20pm", and the 24-hour time 0:00 would be printed as "midnight".

This method should work as follows:

- If the 24-hour time represents midnight or noon, just print "midnight" or "noon".
- Otherwise, if the time is before noon, print it in 12-hour format (ending with "am").
- Otherwise, print the time in 12-hour format, ending with "pm".

Take a look at the `print24HourTime` method, then write the `print12HourTime` method. Also, add a call to `print12HourTime` in `main`, and compile and run the new program.

Part (c). Another useful thing to know is how many minutes there are between two times. For example, it takes 45 minutes to get from noon to 12:45, and it takes 1,395 minutes (23 hours and 15 minutes) to get from 12:45 to noon.

Our next method, `timeRemaining` will tell us how many minutes it takes to get from the time represented by the Time object whose method is called, to a given time. For example, if you have a Time object `now` that represents noon, and another Time object `appointment` that represents 12:45, then the method call `now.timeRemaining(appointment)` will return 45, while the method call `appointment.timeRemaining(now)` will return 1395.

First, figure out (on paper) how to determine the number of minutes it takes to get from one time to another, then write the method.

Add code to `main` to test your method. You may want to do this by reading a time typed in at the terminal, and figuring out how much time there is between that time and the current time (or vice versa). To read one integer from the terminal, use the code on page 5 (*i.e.*, add it to the Time class). You'll also need to put the following line at the beginning of the Time class: `import java.io.*;`

```

import java.util.*;
public class Time
{
    private int hour;
    private int minute;

    public Time(int hr, int min)    // [[ a constructor ]]
    {
        hour = hr;
        minute = min;
    }

    public void print24HourTime()
    {
        System.out.print(hour + ":");
        if (minute < 10) {
            System.out.print("0");
        }
        System.out.println(minute);
    }

    /* main: create a Time object that represents the current tim
    *         and print that time in 24-hour format
    */
    public static void main(String[] args)
    {
        int hour = getCurrHour();
        int min = getCurrMinute();
        Time now = new Time(hour, min);
        now.print24HourTime();
    }

    private static int getCurrHour()
    {
        Calendar cal = new GregorianCalendar();
        return cal.get(Calendar.HOUR_OF_DAY);
    }

    private static int getCurrMinute()
    {
        Calendar cal = new GregorianCalendar();
        return cal.get(Calendar.MINUTE);
    }
}

```

Exercise 4: Syntax

For this exercise, you will divide into groups of 2 or 3 to play *concentration*, which will help you review the syntax for different Java constructs.

Each group will get one set of cards; the green ones have an English description of some kind of Java code, the yellow ones have a definition of the syntax for that code, and the pink ones have an example of that kind of code. First, look at the cards and decide which ones match (note that multiple cards from one group may match a single card from another group).

Now put all the cards face down and take turns turning over two cards of different colors at a time. If you turn over matching cards, you take them and go again. The game ends when someone has 4 matching pairs.

(The cards to be used for this exercise can be created by printing the last three pages of this document and then cutting them up. The first page should be printed on green paper, the second on yellow, and the third on pink.)