INF100 H17 Assignment 3

Due on Friday, September 15, 2017

Albin Severinson, Dag Haugland

Abstract

For this third INF100 assignment, we will write be playing around with dice. Specifically, we will simulate dice rolls until they come up according to some specified outcome. As before we will only grade (pass/fail) the program for the final problem of the assignment. If you have questions specific to this assignment you should send them to me at `albin.severinson@uib.no`.

Submission

1. Add your program file to a `.zip` archive named `ex3_firstname_lastname.zip`. You do this by right clicking on the file ending with `.java` that contains your program code and choosing compress or add to archive (exactly what it is called varies by operating system and version). You should only add your final program, i.e., the one for problem 3. Ask your group leader if you are having issues with this.
2. Log in to `https://mitt.uib.no` and press the box marked with "INF100" to enter the page for this course.
3. Press "Oppgåver" on the left. Next, select "Assignment 3".
4. Press "Lever oppgåve" and upload the `.zip` file you created in step 1.

General Guidelines

These are general guidelines that will (hopefully) help you maintain your sanity through your programming endeavours.

- Have your computer indent your code for you. You do this in DrJava by selecting all of your code, right clicking, and selecting `indent lines`.
- You should make a copy of all your code after completing each problem so that you can roll back to a previous version when you inadvertently break something. Even the developers at Google sometimes have to roll back.
- Set aside time to clean up your code. The number that gets bandied around on the internet is that the best programmers spend roughly half their time cleaning up and rewriting their code, i.e., you are halfway done when you get it working.
- You will save time by making sure you properly understand your current program before moving on to the next problem.
- You should probably be spending a significant part of your time on stackoverflow when working on these assignments.
- If you want to figure out how a standard class or method works (`Scanner`, say), you can look it up on the Java docs page. Instead of trying to find what you are looking for at the Java docs webpage you should probably just Google it (searching for "java Scanner" on Google gives you its docs page as the first result).

Problem 1

Write a program that throws a die until it shows a value equal to a certain target value. The target value should be input to the program from the keyboard. If a target outside the set $1, \ldots, 6$ is given, an error message should be printed. You should print the outcome of each die roll. Here are two tips to get you started:

- You can generate a random number in the set $1, \ldots, 6$ with the code `(int) (1 + 6 * Math.random());`
- You can exit the program prematurely with `System.exit(1);`

Example

Your program output should look like this.

```
Enter die target (1-6):
> 0
Invalid target
```

```
Enter die target (1-6):
> 7
Invalid target
```

```
Enter die target (1-6):
> 5
rolled (6)
rolled (1)
rolled (1)
rolled (4)
rolled (6)
rolled (6)
rolled (5)
got the target outcome in 7 rolls
```

Problem 2

Extend your program to roll two dice at a time. The valid targets are thus $2, \ldots, 12$. For each throw, print the outcome of both dice as well as the sum of the two outcomes. The program should otherwise be the same as for the previous problem.

Example

Your program output should look like this.

```
Enter die target (2-12):
> 1
Invalid target
```

```
Enter die target (2-12):
> 13
Invalid target
```

Problem 2 [Example] continued on next page...

```
Enter die target (2-12):
> 8
rolled (1, 6) SUM=7
rolled (2, 1) SUM=3
rolled (6, 1) SUM=7
rolled (5, 2) SUM=7
rolled (3, 6) SUM=9
rolled (3, 5) SUM=8
got the target outcome in 6 rolls
```

Problem 3

This problem is identical to Problem 2, with the only difference that the target value must be reached a specified number of times. This number should also be input from the keyboard. Furthermore, the program should print an error message if the number of required matches is below 1. Note that putting a `while` loop within another `while` loop is perfectly fine.

Example

Your program output should look like this.

```
Enter die target (2-12):
> 8
Enter number of required matches:
> 0
Invalid number of matches
```

```
Enter die target (2-12):
> 8
Enter number of required matches:
> 2
rolled (5, 6) SUM=11
rolled (1, 6) SUM=7
rolled (6, 4) SUM=10
rolled (3, 3) SUM=6
rolled (3, 2) SUM=5
rolled (1, 2) SUM=3
rolled (1, 4) SUM=5
rolled (5, 4) SUM=9
rolled (3, 1) SUM=4
rolled (1, 1) SUM=2
rolled (4, 1) SUM=5
rolled (3, 5) SUM=8
rolled (3, 6) SUM=9
rolled (6, 2) SUM=8
got the target outcome 2 times in 14 rolls
```

Getting Ahead

This was perhaps your first time programming with loops. If you have completed the assignment, well done! There is still an improvement we can make though.

This part is meant as preparation for the coming course material and will not be graded. Then again, you never have to catch up if you are always getting ahead.

Methods

So far we have been placing all of the code within `main(String[] args) { ... }`. The cleaner way it to use methods. Here is a short introduction: `https://www.youtube.com/watch?v=-IJ5izjbWIA`. Write a separate method that returns a random value in the range $1, \ldots, 6$ (remember to make a copy first!). This avoids having to repeat code.