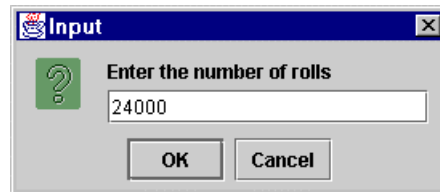


Lecture 5

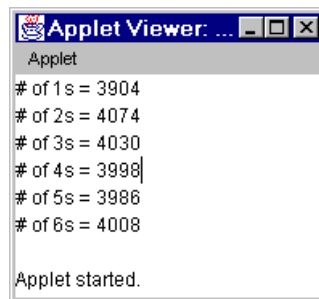
In this lecture we shall discuss the technique of constructing user-defined methods in a class. The discussion will be centered about an experiment of tossing a die a specified number of times.

A die is a six sided cube having 1 to 6 spots on its six edges. When an honest die is rolled, say, 240 times a "1" should appear about $1/6$ of the time; that is, a "1" should come up about 40 times. The same can be said for each of the other faces. In 240 tosses each face should appear about 40 times.

The first applet we shall build will ask the user how many times the die is to be tossed. The request is made by utilizing a JOptionPane input form:



In this example the user typed in 24000. After the OK button the applet simulates 24000 rolls of a die and then presents summary of the results:



In this experiment 3904 1s were rolled. Each face appears "about", not "exactly", 4000 times.

Rolling a Die

The Math method `random()` can be used to simulate any number of rolls of a die. The java call

`Math.random()`

produces a "random" number that is greater than or equal to 0 and strictly less than 1. Intuitively, if we called `random()` 1000 times we should get a thousand numbers evenly spread out in the interval $[0, 1)$.

The java command

`6*Math.random()`

will produce a random double in the range from 0 to 5.99999... The command

```
(int) (6*Math.random())
```

will produce a random integer in the range from 0 to 5. This is because casting, wrapping a pair of parentheses about int, returns the integer part of the double number that follows (int). Finally

```
1+(int) (6*Math.random())
```

will return a random integer between 1 and 6.

Example: Simulating Tossing a Die, An Overview

The structure of the applet to be constructed might be outlined as: The user is prompted to enter the number of rolls. Then the specified number of rolls are made and the results summarized.

This is an outline of the code to be written:

```
import java.awt.*;
import javax.swing.*;

public class TossDie extends JApplet
{
    public void init()
    {
        setSize(200,100);

        Container c = getContentPane();
        JTextArea outputArea = new JTextArea();
        c.add(outputArea);

        int M;
        String Tabulation;

        M = getNumberOfRolls();
        Tabulation = rolloften(M);
        outputArea.setText(Tabulation);

    } // end init()

    public int getNumberOfRolls()
    {
    } // end getNumberOfRolls()

    public int rollOnce()
    {
    } // end rollOnce()

    public String rolloften(int rolls)
    {
    } // rolloften(int rolls)

} // end class TossDie
```

Three methods are to be constructed in this applet:

```
public int getNumberOfRolls()  
public int rollOnce()  
public String rollOften(int rolls)
```

The syntax for the first line of a method is

the word **public** (in all three cases here and in the near future);
the data type the method will return
(the first two will return the type **int**, the third **String**);
the name of the method (it should be descriptive).

The final pair of parentheses **()** in the title line is part of the name and is called the parameter list of the method. In the first two cases the parameter list is empty. In the last case the parameter list **(int rolls)** states what parameters(s) the method will need to carry out its' job. The parameter's type as well as its name appears; no commas separate the two.

Once the methods are constructed they are strung together in the method **init()** to carry out the task at hand. The code

```
setSize(200,100);  
  
Container c = getContentPane();  
JTextArea outputArea = new JTextArea();  
c.add(outputArea);
```

sets the size of the output screen and provides a text area and container for the text. Next

```
M = getNumberOfRolls();
```

gets the number of times the user wants to toss the die. The method **rollOften(M)** does two things: it rolls the die M times and summarizes the results in a string, which is captured by the string **Tabulation**.

```
Tabulation = rollOften(M);
```

The summary is printed out by means of the line:

```
outputArea.setText(Tabulation);
```

Constructing the Methods

We begin by writing **getNumberOfRolls()**. It is built in a straightforward way, utilizing **JOptionPane**:

```
public int getNumberOfRolls()  
{  
    String numberOfRolls;  
    int m;  
  
    numberOfRolls =  
        JOptionPane.showInputDialog
```

```

        ("Enter the number of rolls");
        m = Integer.parseInt(numberOfRolls);

        return m;
    } // end getNumberOfRolls()

```

Note the last line of the method, `return m;`. The value a method returns appears in the last line of the method and is preceded by the word "return".

A method need not "return" a value. If this is the case the type of the method is replaced by the word "void". The first line of the `init()` method

```

        public void init()

```

is an example of such a method. We will come back to this point later.

The method `rollOnce()` simulates one toss of a die.

```

    public int rollOnce()
    {
        int face;
        face = 1 +(int)(6*Math.random());
        return face;
    } // end rollOnce()

```

The value returned here is an integer between 1 and 6.

Finally `rollOften(int rolls)` executes, summarizes, and returns the summary as a string:

```

    public String rollOften(int N)
    {
        int face1 = 0, face2 = 0, face3 = 0,
            face4 = 0, face5 = 0, face6 = 0;

        String output = "";

        int result;
        for (int i = 1; i <= N; i++)
        {
            result = rollOnce();
            if (result == 1) face1++;
            if (result == 2) face2++;
            if (result == 3) face3++;
            if (result == 4) face4++;
            if (result == 5) face5++;
            if (result == 6) face6++;
        } //end for

        output = output + " # of 1s = " + face1 + "\n"+
            " # of 2s = " + face2 + "\n"+
            " # of 3s = " + face3 + "\n"+
            " # of 4s = " + face4 + "\n"+
            " # of 5s = " + face5 + "\n"+
            " # of 6s = " + face6 + "\n";
    }

```

```
return output;
```

```
}// end rollOften(int rolls)
```

As for the details of `rollOften(int N)`, it is a method whose purpose is to return the string named `output`.

To go on to the details, we assign the initial values of 0 to `face1` through `face6` and the null value "" to the string `output`:

```
int face1 = 0, face2 = 0, face3 = 0,  
    face4 = 0, face5 = 0, face6 = 0;
```

```
String output = "";
```

If you do not assign these initial values then, at compile time, you will get an error message saying that these values need to be initialized.

To continue, the for loop

```
for (int i = 1; i <= N; i++)  
{  
    int result = rollOnce();  
    if (result == 1) face1++;  
    if (result == 2) face2++;  
    if (result == 3) face3++;  
    if (result == 4) face4++;  
    if (result == 5) face5++;  
    if (result == 6) face6++;  
}
```

```
} //end for
```

carries out the experiment N times and keeps a record of how many times each face has been rolled.

Finally, a summary is made and returned as the value of the method:

```
output = output + " # of 1s = " + face1 + "\n"+  
    " # of 2s = " + face2 + "\n"+  
    " # of 3s = " + face3 + "\n"+  
    " # of 4s = " + face4 + "\n"+  
    " # of 5s = " + face5 + "\n"+  
    " # of 6s = " + face6 + "\n";
```

```
return output;
```

The complete program is

```
import java.awt.*;  
import javax.swing.*;  
  
public class TossDie extends JApplet  
{
```

```

public void init()
{
    Container c = getContentPane();
    JTextArea outputArea = new JTextArea();
    c.add(outputArea);

    int M;
    String Tabulation;

    M = getNumberOfRolls();
    Tabulation = rollOften(M);
    outputArea.setText(Tabulation);

} // end init()

public int getNumberOfRolls()
{
    String numberOfRolls;
    int m;

    numberOfRolls =
        JOptionPane.showInputDialog
        ("Enter the number of rolls");
    m = Integer.parseInt(numberOfRolls);
    return m;

} //end getNumberOfRolls()

public int rollOnce()
{
    int face;
    face = 1 +(int)(6*Math.random());
    return face;
} // end rollOnce()

public String rollOften(int N)
{
    int face1 = 0, face2 = 0, face3 = 0,
        face4 = 0, face5 = 0, face6 = 0;

    String output = "";

    for (int i = 1; i <= N; i++)
    {
        int result = rollOnce();
        if (result == 1) face1++;
        if (result == 2) face2++;
        if (result == 3) face3++;
        if (result == 4) face4++;
        if (result == 5) face5++;
        if (result == 6) face6++;

    } //end for

```

```

        output = output + " # of 1s = " + face1 + "\n"+
                           " # of 2s = " + face2 + "\n"+
                           " # of 3s = " + face3 + "\n"+
                           " # of 4s = " + face4 + "\n"+
                           " # of 5s = " + face5 + "\n"+
                           " # of 6s = " + face6 + "\n";

    return output;

} // end rollOften(int rolls)

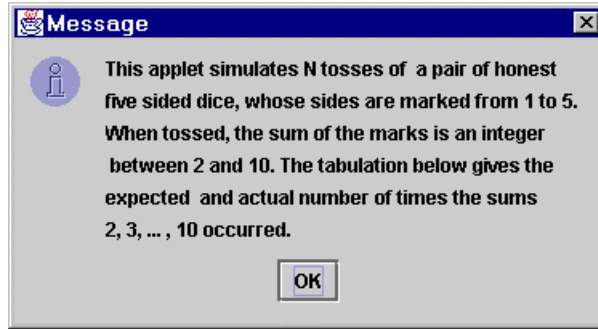
} // end class TossDie

```

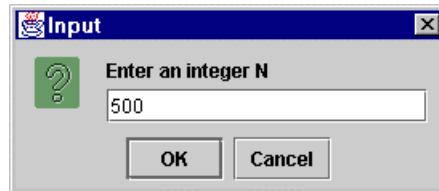
Lab 3.

You are to write an applet that simulates N tosses of a pair of honest, five sided dies. The program, to be called **FiveSidedDie**, combines the looping of Lab2 with the use of methods illustrated in today's lecture. JoptionPanEs are to be used throughout the program for getting and giving information.

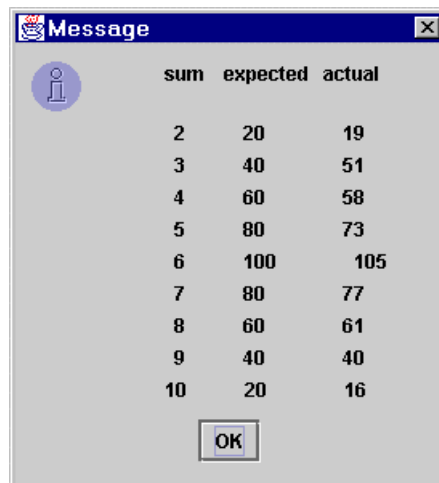
The program is to begin by telling the user what the applet does:



After the user clicks on the OK button a window requesting the integer N . In the window below the user has typed in 500:



After the OK button is clicked here a summary of the expected and actual number of tosses is presented in a JOptionPane:



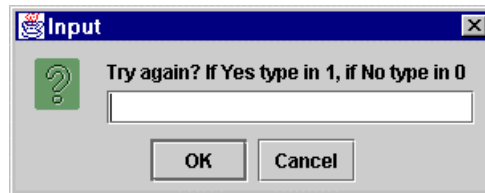
A little bit of probability: since each face of the honest 5-sided die will appear with probability $1/5$ the expected number of times of tossing a sum from 2 to 10 in N tosses may be tabulated as:

| | | | | | | | | | |
|-----|---|---|---|---|---|---|---|---|----|
| sum | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|-----|---|---|---|---|---|---|---|---|----|

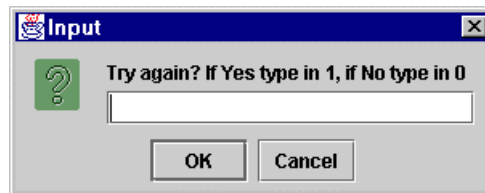
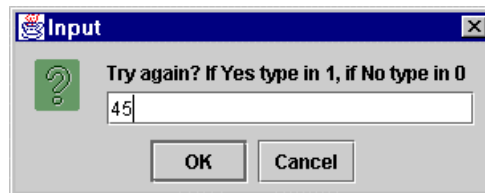
| | | | | | | | | | |
|-------|------|-------|-------|-------|-------|-------|-------|-------|------|
| exptd | N/25 | 2N/25 | 3N/25 | 4N/25 | 5N/25 | 4N/25 | 3N/25 | 2N/25 | N/25 |
|-------|------|-------|-------|-------|-------|-------|-------|-------|------|

Also, use java's division of integers by integers method when presenting these expected number of tosses. (Consequently, the expected values will be correct if the N entered is a multiple of 25, but will be off by as much as 4/5 if N is not a multiple of 25)

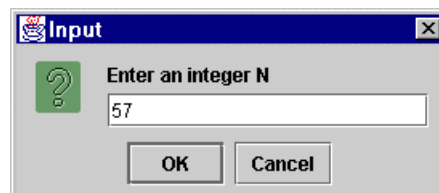
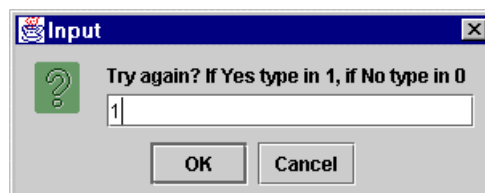
To, continue, once the user clicks the OK button in the last window he or she will be asked if they wish to try another simulation:

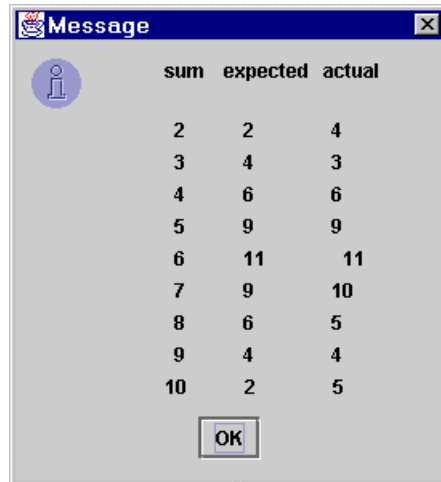


If the integer entered is not a 0 or a 1 the program will cycle:



If the user types in a 1, the program repeats by asking for N:





| | sum | expected | actual |
|----|-----|----------|--------|
| 2 | 2 | 4 | |
| 3 | 4 | 3 | |
| 4 | 6 | 6 | |
| 5 | 9 | 9 | |
| 6 | 11 | 11 | |
| 7 | 9 | 10 | |
| 8 | 6 | 5 | |
| 9 | 4 | 4 | |
| 10 | 2 | 5 | |

The expected column totals up to 53, not 57. This is because of java's method of integer division, mentioned earlier. The actual column does add up to 57, the total number of tosses.