Lecture Seven – Math methods

- GUI review
- Interest Calculator
- Math methods
- Math comparisons
- Math constants
- Formatting numbers
- Roll Dice program
- Random numbers
- Rounding numbers
- Casting
- Drill

Read Chapter 5, pg. 115
GUI review

- In the last lesson we created a simple calculator GUI. We will continue to build from that GUI in this lesson.

- There are still some GUI concepts that we are using but have not gone over in detail. Don’t worry too much about why things work the way they do – we will go over in detail the inner workings of GUIs in future lessons.
GUI review

Recall the major steps we took when we constructed our `SimpleCalc` program:

1. Import library packages
2. Open the class
3. Declare the components
4. Write a ‘main’ method
5. Begin the constructor method
6. Select a layout manager
7. Add components to the layout
8. Add listeners to buttons and the window
9. Write the ‘actionPerformed’ method
10. Complete the ‘WindowListener’
Interest calculator

- Suppose we have just written a Java program and a major software company has purchased it from us for $500,000. We decide we want to invest this money but there are so many options! There are CDs at 4.5%, compounded yearly for 10 years; money market accounts at 4.0%, compounded monthly, etc.

- Our goal is to write a simple interest calculator that we can use to determine how much money we would earn by investing our money in these different types of accounts.
Interest calculator

- Compound interest is determined by the following equation:
  \[ B = P \times (1 + \frac{r}{c})^{c \times n} \]

- Where
  - \( B \) = ending balance
  - \( P \) = initial principal amount invested
  - \( r \) = interest rate
  - \( c \) = number of times amount is compounded per year
  - \( n \) = number of years invested

- Furthermore, we can determine the amount of money we made by subtracting our initial principal from our ending balance \((B - P)\)
We decide we want our interface to look something like this:
Interest calculator

- We will now write our `InterestCalc` using the same steps we took to create our `SimpleCalc`.

- Follow along and (as always) write the program as we go!
Step 1: Import library packages

// For GUI stuff
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

// For Math functions and NumberFormat
import java.lang.Math;
import java.text.*;

- This is just like the SimpleCalc program except we also add libraries for Math and NumberFormat classes
Step 2: Declare the class InterestCalc

// For Math functions and NumberFormat
import java.lang.Math;
import java.text.*;

// Create class
public class InterestCalc extends JFrame
   implements ActionListener, WindowListener
{

Step 3: Declare the components

{
    // Declare GUI components
    private JLabel principalLabel;
    private JTextField principalField;
    private JLabel rateLabel;
    private JTextField rateField;
    private JLabel compoundLabel;
    private JTextField compoundField;
    private JLabel yearsLabel;
    private JTextField yearsField;
    private JLabel balanceLabel;
    private JTextField balanceField;
    private JLabel moneyLabel;
    private JTextField moneyField;
    private JButton calculateButton;
}
Step 4: Write the ‘main’ method

private JButton calculateButton;

// main method
public static void main(String args[]) {
    JFrame frame = new InterestCalc();
    frame.setSize(350, 250);
    frame.setTitle("Interest Calculator");
    frame.setVisible(true);
}
Step 5: Begin the ‘constructor’ method

// constructor
public InterestCalc()
{
    principalLabel = new JLabel(" Initial Principal: ");
    principalField = new JTextField(); // no initial value or width
    rateLabel = new JLabel(" Interest Rate: ");
    rateField = new JTextField();
    compoundLabel = new JLabel(" Compound Times Per Year: ");
    compoundField = new JTextField();
    yearsLabel = new JLabel(" Years: ");
    yearsField = new JTextField();
    balanceLabel = new JLabel(" Ending Balance: ");
    balanceField = new JTextField();
    moneyLabel = new JLabel(" Money Made: ");
    moneyField = new JTextField();
    calculateButton = new JButton("Calculate");
Step 6: Select and set a layout manager

- In the SimpleCalc we used a FlowLayout. For our InterestCalc we will learn about a new layout, GridLayout (see pg 115)
- When creating a new GridLayout, you must pass two arguments to the method. The first is the number of rows in your grid, the second is the number of columns in your grid.
- GridLayout will then divide the frame into a grid and will add components from left to right, top to bottom.
- One thing to note about GridLayout is that components are resized to fit entirely into the grid space allocated for that component. (This is why there was no need to specify a field width when we created our JTextFields in the previous slide.)
Step 6: Select and set a layout manager

... calculateButton = new JButton("Calculate");

// create layout
// grid layout with 7 rows and 2 columns
GridLayout layout = new GridLayout(7, 2);
Container window = getContentPane();
window.setLayout(layout);
Step 7: Add the components to the layout

// add components to window
window.add(principalLabel);
window.add(principalField);
window.add(rateLabel);
window.add(rateField);
window.add(compoundLabel);
window.add(compoundField);
window.add(yearsLabel);
window.add(yearsField);
window.add(balanceLabel);
window.add(balanceField);
window.add(moneyLabel);
window.add(moneyField);
window.add(calculateButton);
Step 8: Add listeners to button and window

\[
\ldots
\]

\[
\text{window.add(calculateButton);}\\
\]

\[
// \text{add action listeners}\\
\text{calculateButton.addActionListener(this);}\\
\text{addWindowListener(this);}\\
\]

\[
} // \text{close constructor}
\]
Step 9: Write the ‘actionPerformed’ method

// action performed method
public void actionPerformed (ActionEvent e) {
    String tempString;
    double principal, rate, compound, balance, years, moneyMade;
    NumberFormat currency = NumberFormat.getCurrencyInstance();
    if (e.getSource() == calculateButton) {
        // get data and convert to doubles
        tempString = principalField.getText().trim();
        principal = (new Double(tempString)).doubleValue();
        tempString = rateField.getText().trim();
        rate = (new Double(tempString)).doubleValue() / 100.0;
        tempString = compoundField.getText().trim();
        compound = (new Double(tempString)).doubleValue();
        tempString = yearsField.getText().trim();
        years = (new Double(tempString)).doubleValue();
    }
Step 9: Write the ‘actionPerformed’ method (cont)

... years = (new Double(tempString)).doubleValue();

// calculate balance and money made
balance = Math.pow(1.0 + rate/compound, compound * years) * principal;
moneyMade = balance - principal;

// output balance and money made in text box as currency
balanceField.setText(currency.format(balance));
moneyField.setText(currency.format(moneyMade));
}
Step 10: Complete the ‘WindowListener’

// window actions
public void windowClosing(WindowEvent e) {
    System.exit(0);
}
public void windowActivated(WindowEvent e) {}
public void windowClosed(WindowEvent e) {}
public void windowDeactivated(WindowEvent e) {}
public void windowIconified(WindowEvent e) {}
public void windowDeiconified(WindowEvent e) {}
public void windowOpened(WindowEvent e) {}
}
} // close InterestCalc class
Interest calculator

- Compile the code and see if it works.
- Investing my money into a 10 year CD at 4.5% interest compounded monthly looks like a good deal!
Math methods

- The InterestCalc program uses a method from the Math class – pow() – which takes 2 arguments: a base and an exponent.
- In order to use methods from the Math class we must import java.lang.Math;
- A list of commonly used Math methods are listed in your textbook on pg. 57.
Math methods (cont)

- Most of the methods listed take a single argument (with the exception of `max()`, `min()`, and `pow()` which all take 2 arguments)
- Most methods also have a return type of type `double`

- Exceptions:
  - `abs()` – returns the same type of value that was used as an argument
  - `max()` & `min()` – returns a value that is the same type as the argument with the most precision (more on this later)
  - `round()` – returns the `int` value that is closest to the `double` passed in as an argument
Two special Math methods – `max()` and `min()` can be used to compare two numbers.

Each method takes two numbers as arguments and will return the number that is either the greatest (max) or the least (min).

The return type is determined by the argument with the greatest precision.

For example,

```java
System.out.println(Math.max(24, 23.0));
// this will print “24.0” NOT “24” since the argument with greatest precision is 23.0, which is a double
```
Math constants

- In addition to the Math methods, there are also 2 Math constants

- These constants are of type double and can be used in programs to perform certain calculations

- The two constants are:
  - Math.PI – value of pi
  - Math.E – value of base of natural logs

- For example, to calculate the area of a circle given a radius, \( r \), would be

\[
\text{area} = \text{Math.PI} \times \text{pow}(r, 2);
\]
Formatting numbers

- In addition to using `Math` methods in the `InterestCalc` program, we also used some methods from the `NumberFormat` class in order to format the output of our balance as currency.

- In order to use `NumberFormat` we need to import `java.text.*;`

- Once we import the library we then need to create a `NumberFormat` instance. In our program we did this with this line of code:

```java
NumberFormat currency = NumberFormat.getCurrencyInstance();
```
Formatting numbers (cont)

```java
NumberFormat currency = NumberFormat.getCurrencyInstance();
```

- This line of code created a currency format.
- We can also create other formats by calling the `getNumberInstance()` or `getPercentInstance()` methods.
  - `getNumberInstance()` will format numbers with added commas (Ex: 1000000 becomes 1,000,000)
  - `getPercentInstance()` will format numbers as percentages (Ex: 0.75 becomes 75%)
Formatting numbers (cont)

- Once we create a format we must then specify which value we want to format.
- In our example we created a format named `currency` and wanted to apply it to our variable named `balance`.
- To do this we called the `format()` method which then returns a `String` that we could display in our `JTextField` `balanceField`.

```java
balanceField.setText(currency.format(balance));
```
Roll Dice program

- Let’s take a look at another example that uses other **Math** methods
- For this example we are going to create a Dice emulator (i.e. a program that mimics the act of rolling dice)
- We will pretend we have a pair of dice and want to generate a random roll each time the Roll button is pressed
- Again, we can modify our previous GUIs to achieve this
- However, we will not go through the 10 steps to create the GUI (you should be familiar with them by now)
// For GUI stuff
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

// For Math functions
import java.lang.Math;

// Create class
public class RollDice extends JFrame implements ActionListener, WindowListener
{
    // Declare GUI components
    private JLabel displayLabel;
    private JButton rollButton;
// main method
public static void main(String args[]) {
    JFrame frame = new RollDice();
    frame.setSize(150, 100);
    frame.setTitle("Roll Dice");
    frame.setVisible(true);
}

// constructor
public RollDice() {
    JLabel displayLabel = new JLabel("Click Roll");
    JButton rollButton = new JButton("Roll");
}
Roll Dice program – code (3)

// create layout
FlowLayout layout = new FlowLayout();
Container window = getContentPane();
window.setLayout(layout);

// add components to window
window.add(displayLabel);
window.add(rollButton);

// add action listeners
rollButton.addActionListener(this);
addWindowListener(this);
}
// action performed method
public void actionPerformed(ActionEvent e) {
    int die1, die2;
    if (e.getSource() == rollButton) {
        die1 = (int) Math.ceil(Math.random() * 6);
        die2 = (int) Math.ceil(Math.random() * 6);
        displayLabel.setText("You rolled a " + die1 + " and a " + die2);
    }
}
// window actions
public void windowClosing(WindowEvent e) {}
{
    System.exit(0);
}

public void windowActivated(WindowEvent e) {}

public void windowClosed(WindowEvent e) {}

public void windowDeactivated(WindowEvent e) {}

public void windowIconified(WindowEvent e) {}

public void windowDeiconified(WindowEvent e) {}

public void windowOpened(WindowEvent e) {}
Random numbers

- Our RollDice program should not have any unfamiliar components in terms of the GUI.
- The only lines of code that should be new to you are:

  ```java
  die1 = (int) Math.ceil(Math.random() * 6);
  die2 = (int) Math.ceil(Math.random() * 6);
  ```

- Let’s take a look at this line of code piece by piece.
Random numbers (cont)

die1 = (int) Math.ceil(Math.random() * 6);

- We’ll start by looking from the inside out.
- The first thing we see is `Math.random() * 6`
- The `random()` method is used to create a random number of type `double` that is between 0.0 and 1.0
- Because we want to generate a number that is a valid die roll we need to multiply our number by 6 which then gives us a random number between 0.0 and 6.0
die1 = (int) Math.ceil(Math.random() * 6);

- The next command we use is Math.ceil() which is a **rounding command** used to round numbers up.
- Since it is impossible to have a die roll of 0 we use this command to round our random number up, thus giving us a new range of random numbers between **1.0** and **6.0**.
- Other rounding commands include floor() (which rounds a number down) and round() (which rounds a number either down or up).
Casting

die1 = (int) Math.ceil(Math.random() * 6);

- The final part of this statement is the (int) part which is called a cast
- Since the result of our Math methods are doubles we must cast the results to an int (because we wouldn’t want a die roll of 5.9)
- Casting is used to convert numbers of one type to another type
- In our example we cast down because we cast a value of type double down to a value of type integer (which has less precision). We can also cast values up.
- When casting down there is the potential for precision and data loss because values may be truncated.
  - For example, casting a double (say 5.9) down to an integer will result in 5 (not 6).
Drill 7

1. Create an angle calculator GUI. That is, allow the user to enter an angle into a field as degrees. Display that value as radians and also give the sin, cos, and tan of the angle. Try using the GridLayout like we did in the InterestCalc. Hint: the trig methods require their arguments be in radians, not degrees.
2. Create a GUI that can find the two roots of a quadratic function. Allow the user to specify values for $a$, $b$, and $c$. For those of you who are rusty on your algebra that means we want to find where

$$ax^2 + bx + c = 0$$

To do this we can solve using the quadratic equation:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$
Drill 7 (cont.)

3. Create a GUI that asks a user to guess a number between 1 and 10. Then, randomly generate a number between 1 and 10 and tell the user whether or not he or she guessed correctly.