Chapter 2: Data and Expressions
Lab Exercises

Uppgifterna markerade med stjärna obligatoriska! Restern friviliga.

Topics	Lab Exercises
Print and println	
String literals	Names and Places
String concatenation	
Escape sequences	Two Meanings of Plus *
Variables	Prelab Exercises
Constants	Area and Circumference of a Circle *
Assignment	Painting a Room *
	Average *
Integers and Floating point	Ideal Weight *
	Time *
Arithmetic Expressions	Lab Grades
Input using the Scanner class	
HTML	Introduction to HTML
Applets	Drawing Shapes *
Graphics	The Java Coordinate System *
Colors	
	Creating a Pie Chart *
	Colors in Java

Names and Places

The goal in this exercise is to develop a program that will print out a list of student names together with other information for each. The *tab character* (an escape sequence) is helpful in getting the list to line up nicely. A program with only two names is in the file *Names.java*.

```
// ***************************************************************
//   Names.java
//
//   Prints a list of student names with their hometowns
//   and intended major
// ***************************************************************

public class Names
{
    // ------------------------
    // main prints the list
    // ------------------------
    public static void main (String[] args)
    {
      System.out.println ();
      System.out.println ("\tName\t\tHometown");
      System.out.println ("\t====\t\t========");
      System.out.println ("\tSally\t\tRoanoke");
      System.out.println ("\tAlexander\tWashington");
      System.out.println ();
    }
}
```

1. Save Names.java to your directory. Compile and run it to see how it works.
2. Modify the program so that your name and hometown and the name and hometown of at least two classmates sitting near you in lab also are printed. Save, compile and run the program. Make sure the columns line up.
3. Modify the program to add a third column with the intended major of each person (assume Sally's major is Computer Science and Alexander's major is Math). Be sure to add a label at the top of the third column and be sure everything is lined up (use tab characters!).

Two Meanings of Plus

In Java, the symbol + can be used to add numbers or to concatenate strings. This exercise illustrates both uses.

When using a string literal (a sequence of characters enclosed in double quotation marks) in Java the complete string must fit on one line. The following is NOT legal (it would result in a compile-time error).

```
    System.out.println ("It is NOT okay to go to the next line
                        in a LONG string!!!");
```

The solution is to break the long string up into two shorter strings that are joined using the *concatenation* operator (which is the + symbol). This is discussed in Section 2.1 in the text. So the following would be legal

```
    System.out.println ("It is OKAY to break a long string into " +
                        "parts and join them with a + symbol.");
```

So, when working with strings the + symbol means to concatenate the strings (join them). BUT, when working with numbers the + means what it has always meant—add!

1. **Observing the Behavior of +** To see the behavior of + in different settings do the following:

a. Study the program below, which is in file *PlusTest.java*.

```
// *******************************************************************
//   PlusTest.java
//
//   Demonstrate the different behaviors of the + operator
// *******************************************************************

public class PlusTest
{
    // ---------------------------------------------------
    // main prints some expressions using the + operator
    // ---------------------------------------------------
    public static void main (String[] args)
    {
      System.out.println ("This is a long string that is the " +
                          "concatenation of two shorter strings.");

      System.out.println ("The first computer was invented about" + 55 +
                          "years ago.");

      System.out.println ("8 plus 5 is " + 8 + 5);

      System.out.println ("8 plus 5 is " + (8 + 5));

      System.out.println (8 + 5 + " equals 8 plus 5.");
    }
}
```

b. Save PlusTest.java to your directory.
c. Compile and run the program. For each of the last three output statements (the ones dealing with 8 plus 5) write down what was printed. Now for each explain why the computer printed what it did given that the following rules are used for +. Write out complete explanations.
   - If both operands are numbers + is treated as ordinary addition. (NOTE: in the expression a + b the a and b are called the operands.)
   - If at least one operand is a string the other operand is converted to a string and + is the concatenation operator.
   - If an expression contains more than one operation expressions inside parentheses are evaluated first. If there are no parentheses the expression is evaluated left to right.
d. The statement about when the computer was invented is too scrunched up. How should that be fixed?

Prelab Exercises

1. What is the difference between a variable and a constant?

2. Explain what each of the lines below does. Be sure to indicate how each is different from the others.
   a. int x;

   b. int x = 3;

   c. x = 3;

3. The following program reads three integers and prints the average. Fill in the blanks so that it will work correctly.

```
// *******************************************************************
//   Average.java
//
//   Read three integers from the user and print their average
// *******************************************************************

import java.util.Scanner;
public class Average
{
  public static void main(String[] args)
  {
    int val1, val2, val3;
    double average;
    Scanner scan = new Scanner(System.in) ;

    // get three values from user
    System.out.println("Please enter three integers and " +
                       "I will compute their average");


    //compute the average



    //print the average



  }
}
```

Area and Circumference of a Circle

Study the program below, which uses both variables and constants:

```
//**********************************************************
//  Circle.java
//
//  Print the area of a circle with two different radii
//**********************************************************

public class Circle
{
    public static void main(String[] args)
    {
     final double PI = 3.14159;

     int radius = 10;
     double area = PI * radius * radius;

     System.out.println("The area of a circle with radius " + radius +
                        " is " + area);

     radius = 20;
     area = PI * radius * radius;

     System.out.println("The area of a circle with radius " + radius +
                        " is " + area);

    }
}
```

Some things to notice:

- The first three lines inside *main* are declarations for PI, radius, and area. Note that the type for each is given in these lines: *final double* for PI, since it is a floating point constant; *int* for radius, since it is an integer variable, and *double* for area, since it will hold the product of the radius and PI, resulting in a floating point value.
- These first three lines also hold initializations for PI, radius, and area. These could have been done separately, but it is often convenient to assign an initial value when a variable is declared.
- The next line is simply a print statement that shows the area for a circle of a given radius.
- The next line is an assignment statement, giving variable radius the value 20. Note that this is not a declaration, so the *int* that was in the previous radius line does not appear here. The same memory location that used to hold the value 10 now holds the value 20—we are not setting up a new memory location.
- Similar for the next line—no *double* because area was already declared.
- The final print statement prints the newly computed area of the circle with the new radius.

2. Modify your program as follows:
   - At the very top of the file, add the line

     ```
     import java.util.Scanner;
     ```

     This tells the compiler that you will be using methods from the Scanner class. In the main method create a Scanner object called ***scan*** to read from System.in.
   - **Instead of initializing** the radius in the declaration, **just declare it without giving it a value**. Now add two statements to read in the radius from the user:
     - A *prompt*, that is, a print statement that tells the user what they are supposed to do (e.g., "Please enter a value for the radius.");

- A read statement that actually reads in the value. Since we are assuming that the radius is an integer, this will use the **nextInt()** method of the Scanner- object (called **scan** in your case).

- When the radius gets it second value, make it be twice the original value.
- Compile and run your program.

Painting a Room

File *Paint.java* contains the partial program below, which when complete will calculate the amount of paint needed to paint the walls of a room of the given length and width. It assumes that the paint covers 95 square meter per liter

```
//*****************************************************************
//File: Paint.java
//
//Purpose: Determine how much paint is needed to paint the walls
//of a room given its length, width, and height
//*****************************************************************
import java.util.Scanner;

public class Paint
{
    public static void main(String[] args)
    {
        final int COVERAGE = 12;  //paint covers 12 sq meter/liter
        //declare integers length, width, and height;
        //declare double totalSqM;
        //declare double paintNeeded;
        //declare and initialize Scanner object

        //Prompt for and read in the length of the room

        //Prompt for and read in the width of the room

        //Prompt for and read in the height of the room

        //Compute the total square meter to be painted--think
        //about the dimensions of each wall

        //Compute the amount of paint needed

        //Print the length, width, and height of the room and the
        //number of liters of paint needed.
    }
}
```

Save this file to your directory and do the following:

1. Fill in the missing statements (the comments tell you where to fill in) so that the program does what it is supposed to. Compile and run the program and correct any errors.
2. Suppose the room has doors and windows that don't need painting. Ask the user to enter the number of doors and number of windows in the room, and adjust the total square meter to be painted accordingly. Assume that each door is 2 square meter and each window is 1.5 square meter.

Average

Write a application that reads two integers (using a Scanner object) and prints their average.
(Remember the talk about type casting)

Ideal Weight (BMI)

Write a program to compute the BMI for both males and females. According to one study, the BMI is counted as: weight(kg) / length* length (m) . For example, the BMI for a person 70 kg and 1,68 cm will be about 25 (70/ 1.68*1.68) . Your program should ask the user to enter his/her height and his/her weight. It should then compute and print the BMI. The general outline of your main function would be as follows:

- Declare your variables (think about what variables you need—you need to input two pieces of information (what?), then you need some variables for your calculations (see the following steps)
- Get the input (height in meter) from the user
- Compute the BMI
- Print the answers

Plan your program, then type it in, compile and run it. Be sure it gives correct answers.

Enhance the Program a Bit If you get a BMI between 20 and 25 you have a normal weight. Så extend your program to compute the ideal weigth. Let the BMI value to be 25 and then calculate the idel weight.
Compute the diference between her/his weight and the ideal weight. Print a message to give the user advice: **“ You have to lose x kg ”.**

Time

a) Write an applicaton that reads values representing a time duration in hours minutes and seconds, and then prints the equivalent total number of seconds.
b) Create a revised version of the previous program that reverse the computation. That is, read a value representing the number of seconds, than print the equivalent amount of time as a combination of hours, minutes and seconds. For example, 9999 seconds is 2 hours, 46 minutes and 39 seconds .

Introduction to HTML (Obs! You don’t have to do this exercise if you don’t want, but you need to now how to integrate java applets and web pages)

HTML is the HyperText Markup Language. It is used to describe how text, images, and multimedia are displayed by Web browsers. In this lab you will learn a little about HTML so that you can create a web page containing headings, interesting fonts, lists, and links as well as applets.

HTML uses *tags* to describe the layout of a document; the browser then uses these tags to figure out how to display the document. Tags are enclosed in angle brackets. For example, <title> is a tag that indicates that this section contains the title of the document. Many tags, including <title>, have corresponding end tags that indicate where the section ends. The end tags look just like the start tags except that they start with the character /, e.g., </title>. So the following text indicates that the title of the document is `Introduction to HTML`:

```
<title>Introduction to HTML</title>
```

There are a few tags that almost every document will have: <html>, <head>, <title>, and <body>. Here is an example of a simple HTML document:

```
<HTML>
  <HEAD>
    <TITLE>Introduction to HTML</TITLE>
  </HEAD>

  <BODY>
  In this lab you will learn about HTML, which is lots of fun
  to use.  In particular, you will learn how to use fonts,
  paragraphs, lists, links and applets in a web page.  Now you
  can make your own web page for your friends to visit!
  </BODY>
</HTML>
```

To see what this looks like, open the file in the web browser. Change the size of the browser window (click and drag any corner) and see how the text is reformatted as the window changes. Note that the title appears on the window, not as part of the document.

The HEAD of a document (everything between <HEAD> and </HEAD>) contains the introduction to the document. The title goes in the head, but for now we won't use the head for anything else. The BODY of a document (everything between <BODY> and </BODY>) contains everything that will be displayed as part of the document. Both the HEAD and the BODY are enclosed by the HTML tags, which begin and end the document.

This document contains only plain text, but an HTML document can have much more structure: headings, paragraphs, lists, bold and italic text, images, links, tables, and so on. Here is a document containing a heading, two paragraphs, and some fancy fonts:

```
<HTML>
  <HEAD>
    <TITLE>Introduction to HTML</TITLE>
  </HEAD>

  <BODY BGCOLOR="lightgreen">
  <H1 align="center">Introduction to HTML</H1>

  <P>In this lab you will learn about <I>HTML</I>, which
  is lots of fun
  to use.  In particular, you will learn how to use fonts,
  paragraphs, lists, links, and colors in a web page.  Now you
  can make your <B>own</B> web page for your friends to visit!</P>

  <P>Later in this lab you will do some fancier stuff with
  applets and graphics and include an applet on your web page.
  Can't you just feel the job offers start rolling in?</P>
  <U>Yippee!</U>
  </BODY>
</HTML>
```

Run the HTML document to see what this looks like in the browser.

In this document the <H1> tag creates a level 1 heading. This is the biggest heading; it might be used at the beginning of the document or the start of a new chapter. Level 2 through level 6 headings are also available with the <H2> through <H6> tags.

The <P> tag creates a new paragraph. Most browsers leave a blank line between paragraphs. The <B> tag creates bold text, the <I> tag creates italic text, and the <U> tag creates underlined text. Note that each of these tags is closed with the corresponding end tag. The BGCOLOR attribute on the BODY tag sets the background color.

Note that line breaks and blank lines in the HTML document do not matter—the browser will format paragraphs to fit the window. If it weren't for the <P> tag, the blank line between the paragraphs in this document would not show up in the displayed document.

Exercise #1: For a file to be visible on the Web, it must be where the web server knows how to find it. *** Instruct students how to create and/or access the public html directory on the local system. ***

Open a new file called MyPage.html in a directory where it will be accessible from the web. Write a simple web page about things that interest you. Your page should contain at least the following:

- A title (using the <TITLE> tag)
- Two different levels of headings
- Two paragraphs
- Some bold, italic, or underlined text

Your name should appear somewhere in the document.

When you are done, view your document from the browser. Just type in the URL, don't use File | Open Page.

More HTML (Optional)

Lists We often want to add a list to a document. HTML provides two kinds of lists, *ordered* (e.g., 1, 2, 3) and *unordered* (e.g., bulleted). A list is introduced with the <OL> or <UL> tag, depending on whether it is ordered or unordered. Each list item is introduced with a <LI> tag and ended with the </LI> tag. The entire list is then ended with </OL> or </UL>, as appropriate. For example, the code below creates the list shown; replacing the <OL> and </OL> tags with <UL> and </UL> would produce the same list with bullets instead of numbers.

```
Things I like:
<OL>
<LI>chocolate
<LI>rabbits
<LI>chocolate rabbits
</OL>
```

Things I like:

1. chocolate
2. rabbits
3. chocolate rabbits

Exercise #2: Add a list, either ordered or unordered, of at least three elements to your document.

Links Links connect one document to another. Links are created in HTML with the <A> (anchor) tag. When creating a link you have to specify two things:

- The URL of the document to go to when the link is clicked. This is given as the HREF attribute of the A element.
- How the link should be displayed (that is, what text or image to click on to go to the linked document). This appears between the <A> and </A> tags.
  For example, the code below creates the link shown, which goes to a page about the history of computing:

```
Learn more about <A HREF="http://ei.cs.vt.edu/~history">the history of
computing.</A>
```

```
Learn more about the history of computing.
```

Exercise #3: Add at least one link that ties in to the material on your page.

Drawing Shapes

The following is a simple applet that draws a blue rectangle on a yellow background.

```
// ****************************************************************
//   Shapes.java
//
//   The program will draw two filled rectangles and a
//   filled oval.
// ****************************************************************

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

public class Shapes extends JApplet
{
    public void paint (Graphics page)
    {
      // Declare size constants
      final int MAX_SIZE = 300;
      final int PAGE_WIDTH = 600;
      final int PAGE_HEIGHT = 400;

      // Declare variables
      int x, y;    // x and y coordinates of upper left-corner of each shape
      int width, height; // width and height of each shape

      // Set the background color
      setBackground (Color.yellow);

      // Set the color for the next shape to be drawn
      page.setColor (Color.blue);

      // Assign the corner point and width and height
      x = 200;
      y = 150;
      width = 100;
      height = 70;

      // Draw the rectangle
      page.fillRect(x, y, width, height);
    }
}
```

Study the code, noting the following:

- The program imports javax.swing.JApplet because this is an applet, and it imports java.awt.* because it uses graphics.
- There is no *main* method—instead there is a *paint* method. The paint method is automatically invoked when an applet is displayed, just as the main method is automatically invoked when an application is executed.
- Most of the methods that draw shapes (see the list in Figure 2.12) require parameters that specify the upper left-hand corner of the shape (using the coordinate system described in Section 2.9) and the width and height of the shape. You can see this in the calls to *fillRect*, which draws a rectangle filled with the current foreground color.
- This applet will be drawn assuming the window for drawing (the Graphics object - named *page* here) is 600 pixels wide and 400 pixels high. These numbers are defined in constants at the beginning of the program. (They currently have no use but you will use them later). The width and height of the applet are actually specified in the HTML file that instructs the Web browser to run the applet (remember applets are executed by Web browsers and Web browsers get their instructions from HTML documents—note that the code executed by the browser is the *bytecode* for the program, the *Shapes.class* file). The code in the HTML document is as follows:

```
<html>
<applet code="Shapes.class" width=600 height=400>
</applet>
</html>
```

Save the files *Shapes.java* and *Shapes.html* to your directory. Now do the following:

1. Compile Shapes.java, but don't run it—this is an applet, so it is run through a browser or a special programs.

2. Run the program through the jGrasp by usig **" the strawberry button "** which is actually the **"run applet for curret file button " .**

3. **Applet viewer** is a fri software developed by Sun and is a part of Javas SDK you already downloaded. Now run the program through the Appletviewer by typing the command in the **Command Promt window (At Start/Programs/Accessories)**
 Go to your directory you saved the files Shapes.java and Shapes.html by typing

```
cd  directory name  and than run
appletviewer Shapes.html
```

   You should see a new window open displaying the rectangle.

4. Now change the width to 200 and the height to 300. Save, recompile and run to see how this affects the rectangle.

5. Change x to 400, y to 40, width to 50 and height to 200. Test the program to see the effect.

6. Modify the program so that it draws four rectangles in all, as follows:
   - One rectangle should be entirely contained in another rectangle.
   - One rectangle should overlap one of the first two but not be entirely inside of it.
   - The fourth rectangle should not overlap any of the others.

8. One last touch to the program ... Change the colors for at least three of the shapes so the background and each of the three shapes are different colors (a list of colors is in Figure 2.10 of the text). Also change two of the *fillRect* methods to *fillOval* so the final program draws two rectangles and two ovals. Be sure that the overlap rules are still met.

The Java Coordinate System

The Java coordinate system is discussed in Section 2.7 & 2.9 of the text. Under this system, the upper left-hand corner of the window is the point (0,0). The X axis goes across the window, and the Y axis goes down the window. So the bigger the X value, the farther a point is to the right. The bigger the Y value, the farther it is down. There are no negative X or Y values in the Java coordinate system. Actually, you can use negative values, but since they're off the screen they won't show up!

1. Save files *Coords.java* to your directory. File Coords.java contains an applet that draws a rectangle whose upper lefthand corner is at 0,0. Run this applet in your text editor (the strawberry button).

2. Modify the applet so that instead of 0,0 for the upper lefthand corner, you use the coordinates of the middle of the applet window. This applet is set up to be 600 pixels wide and 400 pixels high, so you can figure out where the middle is. Save, compile, and view your applet. Does the rectangle appear to be in the middle of the screen? Modify the coordinates so that it does appear to be in the middle.

3. Now add four more rectangles to the applet, one in each corner. Each rectangle should just touch one corner of the center rectangle and should go exactly to the edges of the window.

4. Make each rectangle be a different color. To do this, use the setColor method of the Graphics class to change the color (this is already done once). **Do not change the background color once it has been set!** Doing so causes the screen to flicker between colors.

```
// ****************************************************************
//   Coords.java
//
//   Draw rectangles to illustrate the Java coordinate system
//
// ****************************************************************

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

public class Coords extends JApplet
{
    public void paint (Graphics page)
    {
      // Declare size constants
      final int MAX_SIZE = 300;
      final int PAGE_WIDTH = 600;
      final int PAGE_HEIGHT = 400;

      // Declare variables
      int x, y;    // x and y coordinates of upper left-corner of each shape
      int width, height; // width and height of each shape

      // Set the background color
      setBackground (Color.yellow);

      // Set the color for the next shape to be drawn
      page.setColor (Color.blue);

      // Assign the corner point and width and height
      x = 0;
      y = 0;

      width = 150;
      height = 100;

      page.fillRect(x, y, width, height);

    }
}
```

Creating a Pie Chart

Write an applet that draws a pie chart showing the percentage of household income spent on various expenses. Use the percentages below:

Rent and Utilities	35%
Transportation	15%
Food	15%
Educational	25%
Miscellaneous	10%

Each section of the pie should be in a different color, of course. Label each section of the pie with the category it represents—the labels should appear outside the pie itself.

Colors in Java

The basic scheme for representing a picture in a computer is to break the picture down into small elements called *pixels* and then represent the color of each pixel by a numeric code (this idea is discussed in section 1.6 of the text). In most computer languages, including Java, the color is specified by three numbers—one representing the amount of red in the color, another the amount of green, and the third the amount of blue. These numbers are referred to as the *RGB value* of the color. In Java, each of the three primary colors is represented by an 8-bit code. Hence, the possible base 10 values for each have a range of 0-255. Zero means none of that color while 255 means the maximum amount of the color. Pure red is represented by 255 for red, 0 for green, and 0 for blue, while magenta is a mix of red and blue (255 for red, 0 for green, and 255 for blue). In Java you can create your own colors. So far in the graphics programs we have written we have used the pre-defined colors, such as Color.red, from the Color class. However, we may also create our own Color object and use it in a graphics program. One way to create a Color object is to declare a variable of type Color and instantiate it using the constructor that requires three integer parameters—the first representing the amount of red, the second the amount of green, and the third the amount of blue in the color. For example, the following declares the Color object *myColor* and instantiates it to a color with code 255 for red, 0 for green, and 255 for blue.

```
        Color myColor = new Color(255, 0, 255);
```

The statement *page.setColor(myColor)* then will set the foreground color for the page to be the color defined by the *myColor* object. The file *Colors.java* contains an applet that defines *myColor* to be a Color object with color code (200, 100, 255) - a shade of purple. Save the program and its associated HTML file *Colors.html* to your directory, compile and run it using the appletviewer. Now make the following modifications:

1. Change the constructor so the color code is (0,0,0) --- absence of color. What color should this be? Run the program to check.
2. Try a few other combinations of color codes to see what you get. The description of the Color class in Appendix M contains documentation that gives the codes for the pre-defined colors.
3. Notice in the Color class in Appendix M there is a constructor that takes a single integer as an argument. The first 8 bits of this integer are ignored while the last 24 bits define the color—8 bits for red, 8 for green, and the last 8 bits for blue. Hence, the bit pattern

   ```
   00000000000000001111111100000000
   ```

   should represent pure green. Its base 10 value is 65280. Change the declaration of the *myColor* object to

   ```
   Color myColor = new Color (65280);
   ```

   Compile and run the program. Do you see green?

4. The Color class has methods that return the individual color codes (for red, green, and blue) for a Color object. For example,

   ```
   redCode = myColor.getRed();
   ```

   returns the code for the red component of the myColor object (redCode should be a variable of type int). The methods that return the green and blue components are *getGreen* and *getBlue*, respectively. Add statements to the program, similar to the above, to get the three color codes for *myColor* (you need to declare some variables). Then add statements such as

   ```
   page.drawString("Red: " + redCode, ____ , ____ );
   ```

   to label the rectangle with the three color codes (fill in the blanks with appropriate coordinates so each string is drawn inside the rectangle—you also need to set the drawing color to something such as black so the strings will show up). Compile and run the program; for the pure green color above, you should get 0 for red and blue and 255 for green. Now change the integer you use to create the color from 65280 to 115 and see what you get (can you predict?), then try it again with 2486921 (harder to predict!).

```
// ****************************************************************
//   Colors.java
//
//   Draw rectangles to illustrate colors and their codes in Java
// ****************************************************************

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

public class Colors extends JApplet
{
    public void paint (Graphics page)
    {
      // Declare size constants
      final int PAGE_WIDTH = 600;
      final int PAGE_HEIGHT = 400;

      // Declare variables
      int x, y;    // x and y coordinates of upper left-corner of each shape
      int width, height; // width and height of each shape

      Color myColor = new Color (200, 100, 255);

      // Set the background color and paint the screen with a white rectangle
      setBackground (Color.white);
      page.setColor(Color.white);
      page.fillRect(0, 0, PAGE_WIDTH, PAGE_HEIGHT);

      // Set the color for the rectangle
      page.setColor (myColor);

      // Assign the corner point and width and height then draw
      x = 200;
      y = 125;
      width = 200;
      height = 150;

      page.fillRect(x, y, width, height);

    }
}
```

Colors.html

```
<html>
<applet code="Colors.class" width=600 height=400>
</applet>
</html>
```