Lecture 10

- Covers
  - Keyboard input
  - Screen output
  - The Scanner class
  - Documentation and style
- Reading: Savitch 2.3, 2.4

Lecture overview

This lecture

- Has two short sections
  - More on variables & identifiers
  - More on System.out
- Followed by three main sections
  - The Scanner class
  - Programming style
  - The DecimalFormat class

- More on variables and identifiers

Variable declarations

- More than one variable can be declared in a single declaration
- Separate the variables' identifiers in the declaration with a comma

```
int a, b, c;
double d = 4.5, e;
```

Conventions for identifiers

- Class names – start with an uppercase character
- Object, variable, attribute and method names – start with a lowercase character
- Use meaningful names
  - Such as radius, shoeSize and interestRate

- More on System.out

Program I/O

- Program input and output is commonly referred to as I/O
- In this subject, we concentrate on program input from the keyboard and program output to the monitor

System.out

- System.out is an object
- It is an object of class PrintStream
- It has operations such as
  - print
  - println
  - to display information on the console window

System.out

- Both print( ) and println( ) can take one argument – the string to be displayed on the console
- If the argument is a number, it is implicitly converted to a String object

Example

```
System.out.println("Fred");

System.out.print(" Frederica");

System.out.print(42);

System.out.println("Hawaii " + 5 + " O");
```

- Scanner class

Scanner class

- We use the Scanner class for keyboard input in this subject
- It is a new addition to the Java language in order to simplify using keyboard input
- The Scanner class has instance methods that can read in or check the different types of primitive data and String data entered at the keyboard

Scanner class

- In order to use the Scanner class, we first have to tell the compiler where to look for it with the import command

  ```
  import java.util.*;
  ```

- Then, declare and instantiate a Scanner object

  ```
  Scanner keyboard =  new Scanner(System.in);
  ```

- The instance methods defined by the Scanner class can now be used in the keyboard object

Scanner class

- nextInt( ) and nextDouble( )
- The above methods are used to read in specific data types
- To use these methods you need to be sure that the user input is either an integer or double respectively
- If the input is not, an error occurs (InputMismatchException) and the program will terminate

Example

- Program to calculate total seconds, given hours, minutes and seconds

```
import java.util.*;
public class TimeConverter
{
   public static void main(String[ ] args)
   {
      Scanner keyboard = new Scanner(System.in);
      System.out.print("Enter the number of hours: ");
      int hours = keyboard.nextInt();
      System.out.print("Enter the number of minutes: ");
      int minutes = keyboard.nextInt();
      System.out.print("Enter the number of seconds: ");
      int seconds = keyboard.nextInt();
      int totalSeconds = seconds + minutes * 60 + hours * 3600;
      System.out.println("The total number of seconds is: " + totalSeconds);
   }
}
```

Example

- Program output

```
Enter the number of hours: 3
Enter the number of minutes: 12
Enter the number of seconds: 36
The total number of seconds is: 11556
```

```
Enter the number of hours: 3.2
Exception in thread "main" java.util.InputMismatchException
```

** Notice that hours is of type integer in the program code, however in the second example here the user's input is of type double.*

Scanner class

- next( ) returns the next token found in the input stream line, up to a white space
- No matter the data type, the token will always be read in as a String

Example

- Program to read in a response from the user as a String

```
public static void main(String[ ] args)
{
    Scanner keyboard = new Scanner(System.in);
    System.out.println("Do you like Science fiction?");
    String response = keyboard.next( );
    System.out.println("Your response was: \"" + response + "\"");
}
```

```
Do you like Science fiction?
yes
Your response was: "yes"
```

Example

- Program to read in a response from the user as a character

```
public static void main(String[ ] args)
{
    Scanner keyboard = new Scanner(System.in);
    System.out.println("Do you like Science fiction?");
    String response = keyboard.next( );
    char theResponse = response.charAt(0);
    System.out.println("Your response was: \"" + theResponse + "\"");
}
```

```
Do you like Science fiction?
yes
Your response was: "y"
```

**notice that only the first character of the String has been displayed*

Scanner class

- nextLine( ) returns the entire line of text entered, up to the newline character
- Problems can occur if there a more than one new line character sequentially
- If a new line character is encountered, it will be read as a token

Example

- Program to read in a given name and family name from the user

```
public static void main(String[ ] args)
{
   Scanner keyboard = new Scanner(System.in);
   System.out.println("Enter your given name and family name");
   String name = keyboard.nextLine();

   System.out.println("Your name is: \"" + name + "\"");
}
```

```
Enter your given name and family name
John Crichton
Your name is: "John Crichton"
```

Class Exercise

- Write a program to calculate the area of a right angle triangle, prompting the user to input the size of the base and height of the triangle

Solution

Scanner class

- hasNext(), hasNextInt() and hasNextDouble()
- These methods return a boolean value
- hasNext() will check if there is another token to be read
- hasNextInt() will check if the next token is of type integer
- hasNextDouble() will check if the next token is of type double

Scanner class

- close()
- When using a Scanner object it is good programming practice to close the Scanner when it is no longer needed
- To create the Scanner object:

```
Scanner keyboard = new Scanner(System.in);
```

- To close the Scanner object:

```
keyboard.close();
```

Input from text files

- The Scanner class can also be used to read in input from a file
- The only difference is the way the Scanner object is declared and instantiated

```
File fileOpen = new File("myFile.txt");
Scanner keyboard = new Scanner(fileOpen);
```

Example

- Read the first three integers from the file "myFile.txt", add them together and output the result to the screen.

- Assume the contents of myFile.txt is as follows:

23 19 2 14 78

Example

```
public static void main(String[]args) throws IOException
{
        File fileToOpen = new File("myFile.txt");
        Scanner myFile = new Scanner(fileToOpen);
        int number1 = myFile.nextInt();
        int number2 = myFile.nextInt();
        int number3 = myFile.nextInt();

        int total = number1 + number2 + number3;

        System.out.println("The sum of the first three numbers is "
                                + total);
        myFile.close();
}
```

Example

- Output of program:

The sum of the first three numbers is 44

- Documentation & programming style

Documentation & Programming Style

- Programs need to be easy to read and understand
- To achieve that, we need
  - Good documentation
  - Clear code
- Quality documentation facilitates effective maintenance
- Clear code $\Rightarrow$ self documenting, self-explanatory

Programming Style

- Good programming style makes good use of
  - Comments
  - White space
  - Indentation
  - Identifiers (e.g. variable names)
  - Named constants

Comments

- Inside the program, documentation usually takes the form of comments
- Comments
  - Explanations about pieces of code
  - Denoted using

```
// double slash
        in line comments the rest of the line
/* slash-star
        comments until a star-slash occurs */
```

line comment

block comment

Comments (Documentation)

- When should you use comments?
  - Beginning of files/classes
  - Before each method
  - In-line comments to explain complicated or unclear pieces of code

Comments (Documentation)

- Frequently when we are developing code, we wish to try out some different statements from the ones currently in the program
- We may not want to lose what we currently have, as we may wish to reinstate those statements later
- Place comments around code to stop it from executing
- Remove commented code for readability when you are sure it is no longer required

White space

- Blanks, tabs, and newline characters are called white space characters
- Except when white space is used to separate keywords and identifiers, it is ignored by the compiler
- White space can be used to make programs easy to read
- Two main uses of white space
  - Indentation
  - Blank lines to separate parts of programs

Indentation

- Makes a program easier to read
- Sets of instructions that are related should be made to look like a group
- A statement within a statement should be indented
- Use braces to group several statements together

Indentation

- Each level of nesting should be indented further than the last
- 3 or 4 spaces is a usual amount to indent
- Keep indentation consistent throughout the program

Example (Programming style)

- Non-indented:

```
int x=10; while(x>0){System.out.println(x);x=x-1;}
```

- Indented:

```
int x = 10;
while (x > 0)
{
    System.out.println(x);
    x = x - 1;
}
```

Named Constants

- Variables

```
int i;
int j = 1;
```

- Constants

```
final double RATE = 0.2;
// RATE  cannot be changed subsequently
```

- Constants, by convention, are usually written in uppercase letters

Static attributes

- Constants are usually defined as static attributes (class attributes)
- A static attribute belongs to the class and is shared by all instances of the class
- Access static attributes

  <class-name>.<static-attribute>

  area = Math.PI * radius * radius;

- DecimalFormat class

DecimalFormat class

- When displaying numeric data, we can use the DecimalFormat class to control the display format
- It can be used to control many features of the display format
- In this section, we use the DecimalFormat class simply to control the number of fractional digits we want to display

Program example

- Write a program that calculates the area of a triangle, given the lengths of the sides
- Carry out the calculation using Heron's formula:

$$Area = \sqrt{s(s-a)(s-b)(s-c)}$$

- where a, b and c are the three side lengths and s is half the perimeter of the triangle

Program example

- Algorithm

  Prompt user for 3 side values for the triangle
  Get the side lengths
  Calculate s
  Calculate area
  Display area

** Refer to Sun's Java website to find out what classes contain which functions*

** Math contains a sqrt function to help in this case*

Example

- Java solution

```
import java.util.*;
public class Triangle
{
   public static void main(String[ ] args)
   {
      Scanner keyboard = new Scanner(System.in);
      System.out.print("Enter the 3 side lengths of the triangle: ");
      double a = keyboard.nextDouble( );
      double b = keyboard.nextDouble( );
      double c = keyboard.nextDouble( );

      double s = (a + b + c)/2;
      double area = Math.sqrt(s * (s - a) * (s - b) * (s - c));

      System.out.println("The area of that triangle is: " + area);
   }
}
```

Example

- Result

Enter the 3 side lengths of the triangle: 5 5 7
The area of that triangle is: 12.497499749949988

- How can we get the program to only print out to 3 decimal places?
- Use the DecimalFormat class

DecimalFormat class

- The DecimalFormat classes allow us to create objects that will format a floating point number in the way we specify
- Each DecimalFormat object has a format method that takes a number and returns a String object that contains the number formatted as per the object's constraints

DecimalFormat class

- How a DecimalFormat object formats a number is set through the constructor when we create the object
- The constructor can take a String argument that specifies the pattern required

Example

- Java solution using DecimalFormat

```
import java.text.DecimalFormat;
import java.util.*;
public class Triangle
{
   public static void main(String[ ] args)
   {
      Scanner keyboard = new Scanner(System.in);
      System.out.print("Enter the 3 side lengths of the triangle: ");
      double a = keyboard.nextDouble( );
      double b = keyboard.nextDouble( );
      double c = keyboard.nextDouble( );
      double s = (a + b + c)/2;
      double area = Math.sqrt(s * (s - a) * (s - b) * (s - c));
      DecimalFormat areaFormat = new DecimalFormat("0.###");
      String areaFormatted = areaFormat.format(area);
      System.out.println("The area of that triangle is: " + areaFormatted);
   }
}
```

Example

- Result

Enter the 3 side lengths of the triangle: 5 5 7
The area of that triangle is: 12.497

- The pattern "0.###" specifies that a 0 should be placed on the lhs of the decimal point if there is no integer part; the ### following the decimal point specifies 3 decimal places to be shown at most
- We can use this object more than once
- The import statement tells the compiler where to look for the definition of the DecimalFormat class

Example

- Using DecimalFormat objects multiple times

```
import java.text.DecimalFormat;
import java.util.*;
public class NumberFormatTest
{
   public static void main(String[ ] args)
   {
       Scanner keyboard = new Scanner(System.in);
      double a = keyboard.nextDouble( );
      double b = keyboard.nextDouble( );
      double c = a / b;
      DecimalFormat formatter = new DecimalFormat("0.###");
      String formatA = formatter.format(a);
      String formatB = formatter.format(b);
      System.out.println("a: " + formatA);
      System.out.println("b: " + formatB);
      System.out.println("c: " + formatter.format(c));
   }
}
```

Next lecture

- The if…else statement
- Boolean expressions