## Variables and Constants (Outsource: 3-5 – 3-8, 3-44)

Data comes in two forms: variables and constants. Data that cannot be changed after a program is compiled is **constant**. Data that can be changed during program execution is **variable**. If you include the statement `System.out.println(1234);` in a Java program, the number 1234 is referred to as a **literal constant** because the value is taken literally at each use.

On the other hand, if you create a variable called *age* and include the statement `System.out.println(age);` within a Java program, then different values might be displayed when the program is executed multiple times, depending on what value is assigned to **`age`** during each execution of the program.

Variables are names that reference memory locations that your program can use to store values. The Java programming language provides for eight **primitive types** of data:

```
boolean          float
byte             int
char             long
double           short
```

A variable declaration includes the following:

- A data type that identifies the type of data that the variable will store
- An identifier that is the variable's name
- An optional assigned value, when you want a variable to contain an initial value
- An ending semicolon

The requirements for a variable name are the same as the requirements for a class name:

- A variable name must begin with a letter of the alphabet, an underscore, or a dollar sign.
- A variable name can contain only letters, digits, underscores, or dollar signs.
- A variable name cannot be a Java programming language reserved keyword, such as *public* or *class*.
- A variable name cannot be one of the following values: **true**, **false**, or **null**.

For example, the variable declaration `double myPay = 12.5;` declares a variable of type double named myPay and assigns it an initial value of 12.5. The statement must end in a semicolon. The equals sign ( `=` ) is the **assignment operator**. The value to the right of the assignment operator is assigned or stored in the variable on the left of the equals sign. An assignment made when you declare a variable is an **initialization**; an assignment made later is simply an **assignment**.

## The int Data Type (Outsource: 3-12, 3-45 – 3-55)

In the Java programming language, you use variables of type **int** to store **integers**, or whole numbers. An integer can hold any whole number value from -2,147,483,648 to 2,147,483,647. When you assign a value to an int variable, you do not type any commas; you type only digits

and an optional plus or minus sign to indicate a positive or negative integer. The five standard arithmetic operators that can be performed on integers are as follows:

| Operator | Description | Example |
|---|---|---|
| + | Addition | 45 + 2, the result is 47 |
| - | Subtraction | 45 - 2, the result is 43 |
| * | Multiplication | 45 * 2, the result is 90 |
| / | Division | 45 / 2, the result is 22 (not 22.5) |
| % | Modulus (remainder) | 45 % 2, the result is 1 |

The program illustrated in Figure 1 displays the message: **I have 4 pair of Fuzzy Bunny Slippers!** A constant is used to hold the number of slippers.

```
public class FuzzyBunnySlippers
{
    public static void main(String args[]) {
        final int slippers = 4;  // slippers is a constant of type int

        System.out.println("I have " + slippers + " pair of Fuzzy Bunny Slippers!");
    }
}
```

Figure 1

The program illustrated in Figure 2 initializes two integer variables, calculates the sum of the two variables and displays the message: **3 + 4 = 7.**

```
public class SumProgram
{
    public static void main(String args[]) {
        int firstNumber = 3;
        int secondNumber = 4;

        int sum = firstNumber + secondNumber;
        System.out.println(firstNumber + " + " + secondNumber + " = " + sum);
    }
}
```

Figure 2

## The double Data Type (Outsource: 3-22)

A **floating-point** number contains decimal positions. The **double** data type is a floating-point number that can hold 14 or 15 significant digits of accuracy. The term **significant digits** refers to the mathematical accuracy of a value. A double can hold any floating point number from $-1.7 * 10^{308}$ to $1.7 * 10^{308}$. The four standard arithmetic operators that can be performed on doubles are as follows:

| Operator | Description | Example |
|---|---|---|
| + | Addition | 45 + 2, the result is 47 |
| - | Subtraction | 45 - 2, the result is 43 |
| * | Multiplication | 45 * 2, the result is 90 |
| / | Division | 45 / 2, the result is 22.5 |

The program illustrated in Figure 3 initializes two integer variables, calculates the sum of the two variables and displays the message: **3.4 + 4.2 = 7.6.**

```
public class SumProgram
{
    public static void main(String args[]) {
        double firstNumber = 3.4;
        double secondNumber = 4.2;

        double sum = firstNumber + secondNumber;
        System.out.println(firstNumber + " + " + secondNumber + " = " + sum);
    }
}
```

Figure 3

## Numeric Type Conversion (Outsource: 3-43)

When you are performing arithmetic with variables or constants of the same type, the result of the arithmetic remains the same type. For example, when you divide two integers the result is an integer. If you perform arithmetic operations with operands of unlike types, the Java programming language chooses a **unifying type** for the result. The Java programming language then **implicitly** (or automatically) converts nonconforming operands to the unifying type. The unifying type is the type of the involved operand that appears first in the following list:

```
double
float
long
int
short
byte
```

You can explicitly (or purposely) override the unifying type imposed by the Java programming language by performing a type cast. Type casting involves placing the desired result type in parentheses followed by the variable or constant to be case. For example, to divide two integers and get a floating-point result you would need to type cast one of the integers into a double:

```
int num1 = 3;
int num2 = 2;

int result = num1 / num2;                    // result is equal to 1
double result = (double) num1 / num2;        // result is equal to 1.5
```

## Constants (Outsource: 3-44)

Constants are exactly like fields except their values cannot be changed. In the field declaration the final modifiers is used.

- Syntax for declaring a constant in a method:
  **final** <dataType> <variableName> = <value>;

- Syntax for declaring a constant for a class:
  **public static final** <dataType> <variableName> = <value>;

- Constant declarations must include an initializing value.

## Scope

A variable's scope is the region of a program within which the variable can be referred to by its simple name. Secondarily, scope also determines when the system creates and destroys memory for the variable.

The location of the variable declaration within your program establishes its scope.

A **member variable (field)** is a member of a class or an object. It is declared within a class but outside of any method or constructor. A member variable's scope is the entire declaration of the class.

There are two types of member variables. **Instance member variables** store the values that are tied to an object. In other words, every object of the class has its own copy of every instance member variable. **Class member variables** store values that are shared by all instances of the class. In other words, every object of the class shares every class member variable. A *class member* is declared by using the *static* modifier.

You declare **local variables** within a block of code. In general, the scope of a local variable extends from its declaration to the end of the code block in which it was declared.

```
public class Perimeter
{
    private static final double PI = 3.14;
    private int diameter = 6.4;  <-------------  // Member variables can be used
    private int perimeter;                       // anywhere within the class

    public static void main(String[] args) {
        Perimeter per = new Perimeter ();
        per.process();
        per.output();
    }

    public void process() {
        double radius = diameter / 2.0;  <-------  // Local variables can be used only
        perimeter = 2 * PI * raidus;               // in the block in which they were
                                                   // created
    }

    public void output() {
        System.out.println("The perimeter of a circle with a diameter of " +
                           diameter + " is " + perimeter + ".");
    }
}
```

## The Scanner Class

An **interactive** program is one that accepts value from the user at **run time**, that is, while the program is executing. Providing values during the execution of a program requires input, and the simplest form of input to use is keyboard entry from the programmer's perspective.

Java has many classes that provide methods to read data from the keyboard. By far, the simplest to use is the **Scanner** class.

The Scanner class is a simple text scanner which can parse primitive types and strings using regular expressions.

A Scanner breaks its input into tokens using a delimiter pattern, which by default matches white space. The resulting tokens may then be converted into values of different types using the various next methods.

For example, the following code allows a user to read a number from the keyboard (System.in):

```
Scanner reader = new Scanner(System.in);
int i = reader.nextInt();
```

Default `InputStream` (Keyboard)

## Method Summary – Scanner class

| | |
|---|---|
| String | **`next`** `()`<br>Finds and returns the next complete token from this scanner. |
| double | **`nextDouble`**`()`<br>Scans the next token of the input as a `double`. |
| int | **`nextInt`**`()`<br>Scans the next token of the input as an `int`. |
| String | **`nextLine`**`()`<br>Advances this scanner past the current line and returns the input that was skipped. |
| Scanner | **`useDelimiter`**`(String pattern)`<br>Sets this scanner's delimiting pattern to a pattern constructed from the specified `String`. |

## The import Statement

The Java language is rich in prewritten classes. These classes are stored in *packages* which are libraries of classes. To be able to use any of these packages in your Java program you must include an **import** statement and identify which package you will be using. For example, if you want to use the `Scanner` class you must include the following statement:

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

**java.util** is the folder where the `Scanner` class is located. `Scanner` is, of course, the class we would be importing. It is also possible to import all the classes within a folder by including a statement like:

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

which would make all the classes in the java.util folder, including the `Scanner` class, available for use.

## Formatting Decimal Values

Formatting the number of decimal places (precision) of floating point numbers can be accomplished by using the `DecimalFormat` Class. To be able to use the ***`DecimalFormat`*** class you must include an import statement that identifies which package you will be using.

```
import java.text.DecimalFormat;
```

To format a floating point number first instantiate a `DecimalFormat` object. The String argument is used to determine the pattern to be used to format the number. For example, to display a real number with two *fixed* decimal places with commas as a grouping separator you would make the following declaration:

```
DecimalFormat df = new DecimalFormat(",###.00");
```

`df` is a `DecimalFormat` object which can be used to format a floating point number to the designated pattern. The following statements:

```
double d = 12345.1;
System.out.println(df.format(d));
```

results in **12,345.10** being displayed on the screen.

## Special Pattern Characters

Many characters in a pattern are taken literally; they are matched during parsing and output unchanged during formatting. Special characters, on the other hand, stand for other characters, strings, or classes of characters. They must be quoted, unless noted otherwise, if they are to appear in the prefix or suffix as literals.

The symbols listed here are used to create string patterns to be used in instantiating a **DecimalFormat** object.

| Symbol | Meaning |
|---|---|
| **0** | Digit |
| **#** | Digit, zero shows as absent |
| **.** | Decimal separator or monetary decimal separator |
| **-** | Minus sign |
| **,** | Grouping separator |
| **E** | Separates mantissa and exponent in scientific notation. *Need not be quoted in prefix or suffix.* |
| **%** | Multiply by 100 and show as percentage |
| **\u00A4** | Currency sign, replaced by currency symbol. If doubled, replaced by international currency symbol. If present in a pattern, the monetary decimal separator is used instead of the decimal separator. |
| **'** | Used to quote special characters in a prefix or suffix, for example, `"'#'#"` formats 123 to `"#123"`. To create a single quote itself, use two in a row: `"# o''clock"`. |

The program illustrated in Figure 4 initializes two *double* variables, calculates the quotient of the two variables and displays the message: **The quotient of 7.3/2.9 is 2.6**.

```
DecimalFormat df = new DecimalFormat("#.0");

double numerator = 7.3;
double denominator = 2.86;

System.out.println("The quotient of " + df.format(numerator) + "/" +
                   df.format(denominator) + " is " + df.format(numerator / denominator));
```

Figure 4

## The import Statement

To use the `DecimalFormat` class you must include the following import statement:

```
import java.text.DecimalFormat;
```

**java.text** is the folder where the `DecimalFormat` class is located. `DecimalFormat` is, of course, the class we would be importing. It is also possible to import all the classes within a folder by including a statement like:

```
import java.text.*;
```

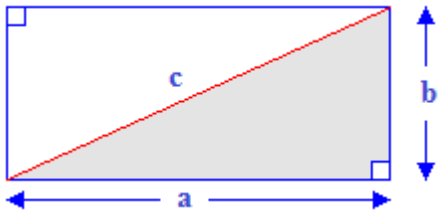
which would make all the classes in the java.text folder, including the `DecimalFormat` class, available for use.

## LAB 02 - ASSIGNMENT

## Lab 04A - 60 points

### OBJECTIVE

WAP that calculates the area of a right triangle given that side **a** is **9** inches and side **b** is **5** inches.




### FIELD SUMMARY

- **int a** – the length of side a.
- **int b** – the length of side b.
- **double area** – the area of the right triangle in cubic inches.

### METHOD SUMMARY

- **main** – instantiate an object of your class. Call `input`, `process` and `output`.
- **input** – declare a `Scanner` object and read the lengths of side a and side b from the keyboard using appropriate prompts.
- **process** – calculate the area of the right triangle.
- **output** – display the results on the console screen. Format all doubles to display one decimal place. Suppress trailing zeros.

### SAMPLE KEYBOARD INPUT

*Enter the length of side a:* **9**
*Enter the length of side b:* **5**

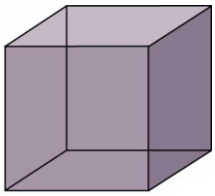
### SAMPLE OUTPUT

**The area of a right triangle with sides of length 9 inches and 5 inches is 22.5 square inches.**

## Lab 04B - 70 points

### OBJECTIVE

WAP that reads the length of one side of a cube from the keyboard and calculates the volume of the cube.



### FIELD SUMMARY

- **double side** – the length of one side of the cube (7.42 inches).
- **double volume** – the volume of the cube.

### METHOD SUMMARY

- **main** – instantiate an object of your class. Call process and output.
- **input** – declare a `Scanner` object and read the length of one side of a cube from the keyboard using an appropriate prompt.
- **process** – calculate the volume of the cube.
- **output** – display the results on the console screen. Format all doubles to display two decimal places. Suppress trailing zeros.

### SAMPLE KEYBOARD INPUT

*Enter the length of one side of a cube:* **7.72**

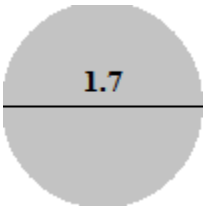
### SAMPLE OUTPUT

**The volume of a cube with a side length of 7.72 inches is 460.1 cubic inches.**

## Lab 02C - 80 points

### OBJECTIVE

WAP that calculates the area of a circle with a diameter of 1.7. Use a constant to represent PI (3.1416). The `radius` and the `area` should both be defined as *fields*. NO NUMERIC VALUES SHOULD BE USED IN THE FORMATION OF THE OUTPUT STATEMENT.




### FIELD SUMMARY

- **final double PI** – PI (defined as 3.1416).
- **double diameter** – the diameter of a circle.
- **double area** – the area of a circle.

### METHOD SUMMARY

- **main** – instantiate an object of your class. Call process and output.
- **input** – declare a `Scanner` object and read the diameter of a circle from the keyboard using an appropriate prompt.
- **process** – calculate the area of the cube.
- **output** – display the results on the console screen. Format all doubles to display two decimal places. Suppress trailing zeros.

### SAMPLE KEYBOARD INPUT

*Enter the diameter of the circle:* **4.25**

### OUTPUT

**The area of a circle with a diameter of 4.25 is 14.19.**

## Lab 02D - 90 points

### OBJECTIVE

- WAP that reads the diameter of a sphere from the keyboard and calculates the *surface area* and *volume* of of the sphere. Use a *constant* to represent PI and *fields* to represent the *diameter*, *volume*, and *surface area*. You must include `input`, `process`, and `output` methods. Format all doubles to display two decimal places. Suppress trailing zeros.

### SAMPLE INPUT

*Enter the radius of the sphere:* **4.25**

### OUTPUT

**The surface area of a circle with a radius of 4.25 is 226.98.**
**The volume of a sphere with radius of 4.25 is 321.56.**

## Lab 02E - 100 points

### OBJECTIVE

- Susan is shopping for a computer system, and is considering three different systems for three different prices. The sales tax on the purchase is 8.25%.
- Define the 8.25% as a constant (hint: change the percent to a real (double) value equivalent), compute and print the total price for one system.
- Input three prices from the keyboard and create three displays like the ones below, one for each system. Format the values to look like money. Make the output appear exactly as shown below.

To solve the scenario described above, WAP that reads the price for a computer system from the keyboard. Calculate the tax and the total cost (price of the computer plus tax). Output the cost of the computer system in the format shown below. Format all doubles to display two decimal places. Show trailing zeros.

### SAMPLE KEYBOARD INPUT

*Enter The Price For The Computer System:* **2145.95**

### SAMPLE OUTPUT

```
SYSTEM COST
===========
Purchase        = $2145.95
Tax             = $177.04
----------
Total           = $2322.99
```