CECS 174 – Lecture 2 – Variables and Console IO

Declaring Variables

Variable – a storage space in memory that holds a value. A variable can only hold one value at a time and that value must be of the declared data type. Once a variable has been declared, it cannot be changed and it will not need to be redeclared.

Data Types –

Integer:

`int` – Size: 4 bytes. Use for whole numbers up to a maximum of 2 billion.
`byte` – Size: 1 byte. For values less than 128.
`short` – Size: 2 bytes. For values smaller than 32768.
`long` – Size: 8 bytes. For very large integer values.

Float:

`double` – Size: 8 bytes. Use for decimal values.
`float` – Size: 4 bytes. Use for less precise decimal values.

Boolean:

`boolean` – Size: 1 byte. True/False values.

Character:

`char` – Size: 2 bytes. Use for single letters, numbers, or symbols. Place the value in single quotes when assigning.

String:

`String` – Used for multiple characters. Place the value in double quotes.

Rules for Naming Variables:

1. Variable names are made up of letters, numbers, underscores and dollar signs, but they cannot begin with a number.
2. Spaces are not permitted within a variable name. Avoid spaces by using an underscore or capitalizing the first letter of the next word.
3. Variable names should start with a lower case letter to distinguish it between constants and class names.
4. You cannot use reserved words as variable names (reserved words are those such as: public, void, final, double, etc.). But they can be used as part of the variable name, such as doubleVal or finalAmount.
5. Variable names are case sensitive. You cannot declare a variable as numPeople and then try to use it as numpeople.
6. Give your variables short, descriptive, and meaningful names, so that when someone is reading through your code they will understand what it is doing.

Example: What data type should each of these variables be?

1. _______ numPeople – stores the number of people counted.
2. _______ employeeHourly – stores the hourly wage of the employee.
3. _______ testDone – stores the result determining if an event occurred.
4. _______ middleInitial – stores the middle initial for a person's middle name.
5. _______ cityName – stores the name of a city.

Example: Are these legal variable names? Why?

1. _______ int x;
2. _______ int numCars;
3. _______ String FirstName;
4. _______ double 2items;
5. _______ char _codeLetter;
6. _______ boolean final;
7. _______ String Last Name;

Assignment Operator - The assignment operator (=) is used to assign the value of the right hand side to the variable on the left. A variable must have a value before it can be used, and it can only be assigned a value if it is of the same data type.

```
String firstName = "John";
String fullName = firstName + " Jones";
int num1 = 5;
int num2 = num1 + 4;
num1 = 3;
double price = 3.99;
```

Constants – Variables whose values you know will not be changed during the run of the program are called constants. Some examples of constants are: the value of Pi, the number of pennies in a dollar, or the percentage of sales tax. Using the keyword `final` when declaring a variable lets the compiler know that you do not want that variable to change. If you do try to change it, the compiler will give you an error.

It is a good formatting practice to capitalize the names of your constants; it helps identify them as constants when others read your code.

Example:

```
final double PI = 3.14159265;
final double SALES_TAX = 0.085;
```

Comments

Comments are used to hide any text or code from the compiler. They are used to make notes about the functionality of your program or to temporarily remove a block of code from your program.

1. Single line comment - `// this is a comment`
2. Multi line comment - `/* this is also a comment`
   `but it's on multiple lines */`

Example: Add some comments:

```
This comment spans multiple lines but has different
options for commenting it out.

String name = "George";    Declares a person's name
```

Input and Output

Outputting Data to the Console Window:

```
System.out.print("Hello There ");
System.out.println(fullName);
```

`print()` does not move the cursor down to the next line after printing, whereas `println()` does create a new line.

Example:

```
public class Program1 {
     public static void main(String[] args) {
          String item = "candy bar";
          int quantity  = 5;
          double price = 0.99;

          System.out.println("You bought " + quantity
          + " " + item + "s. They are $"+ price + "
          each. Your total is: "+ (price*quantity));
     }
}
```

Input – Collecting data from the user – different data types require different inputs.

```
import java.util.Scanner;

public class Program2 {
     public static void main(String [] args) {
          Scanner in = new Scanner(System.in);
          String fName = in.next();
          in.nextLine(); //clears return character
          String item = in.nextLine();
          int quantity = in.nextInt();
          double price = in.nextDouble();

          System.out.println(fName+" bought "+quantity
          +" " +item+ "s. They are $"+ price + " each.
          The total is: "+ (price*quantity));
     }
}
```

Note - next() takes input up to, but not including, a whitespace (space, tab, return) nextLine() takes in input that includes spaces or tabs until the return is pressed.

Example:

Write a program that reads in a number, doubles it, and then displays the result.