CSCI 1301 Lecture 3

**What we will cover:**

Algorithms for Writing Programs
Identifiers, Variables, Literals
Assignment
Constants
Primitive Data Types
Operators
Input and Output

**Algorithms for Writing Programs**

There is no magic to writing a successful program. You should follow the same general steps for writing every program this semester. Hard work, practice, and exposure to code are what elevate programming skill. You should be working with code regularly to develop and keep your programming skills. Even-numbered programming problems in the book have their solutions posted on the book's supplemental website.

**Basic Algorithm for Writing Programs**

1. Identify the problem you are trying to solve
2. Break the problem into distinct pieces
3. Solve each piece of the problem
4. Gather these "small" solutions together into one cohesive program

**Detailed Description for Writing Programs**

Before you code a single thing, stop, and think about the problem at hand. A number of Java programs in this course will contain three major operations: getting input, manipulating that input, and delivering the results via output. Solve the problem on paper or in your head before you open a text editor. Map out what you want the program to do and have a clear picture of how the code should be written to accomplish those tasks. Once that is done, open your editor and create the basic framework of your program (main comment block, class block, and main method). Save often and compile often; if you have an error in your class or main method declaration, this is the best time to find it, not once you’ve written the entire program. If necessary, add comments of the steps you need to complete. Write these small blocks of code and test them to localize errors. Once the program is complete, especially later in the semester, don’t just test cases where you know it will work; test edge cases and boundaries to see how the program will behave in general. Remember that you should be testing these programs as you go. If it feels awkward to use while you are testing it, think how awkward it would feel to someone else using it. Work on presentation just as much as anything else. Not only should the source code look good, the program should look good and act well, too.

**Identifiers, Variables, Literals, Constants**

Identifiers are sequences of characters that are used to refer to something (data, method, etc). Identifiers must start with a letter, $, or underscore and can consist of letters, digits, underscore, and $. Identifiers cannot be keywords or reserved words and can be any length. Identifiers are case sensitive.

A variable is an identifier that refers to a location in the computer’s memory where a value can be stored for use by a program. Variable names must be a legal identifier. The name can be of any length. They should usually start with a lowercase letter and have a meaningful name. If the name is several words concatenated together, all words after the first should have the first letter capitalized.

Before using a variable you need to indicate the type of data it refers to.

```plaintext
<data type> <variable name>;
```

or

```plaintext
<data type> <variable name1>, <variable name2>, ..., <variable name n>;
```
A literal is a constant value that appears directly in the program code. For example: 123 and 23.50 are numeric literals and “Hello” is a string literal. They are often used with variable declarations to provide an initial value, as part of arithmetic expressions, and to create constants.

Note: Variables/identifiers are very useful for any value that could ever change. This includes items like prices, exchange rates, quantities, etc. If you are hard-coding literals into an expression, take a moment and think about whether this expression could ever change, and if so, how hard it might be if a literal is used numerous times in your code. An expression is typically a computation involving literals, variables, and operators that evaluate to a value.

Assignment
After a variable is declared, you can assign it a value using the assignment operator (=). The variable should always be on the left hand side of the equals. The general format for assignment is:

<variable> = <value>;

For example:

    int age;
    age = 23;

You can declare a variable and initialize it in one step:

    int age = 23;

The results of expressions can also be assigned to variables. Consider the second line in the following example;

    double radius = 5;
    double area = radius * radius * 3.14159;

The act of assignment is considered an expression. Remember this when we get to conditional statements and you will understand why there is a different symbol for comparison and assignment.

Constants
Values that do not change during a program (such as π) can be defined as constants. The value of a named constant cannot be changed during program execution. Java convention states that constant names are all in caps. The constant must be declared and initialized in the same statement. The syntax for defining a constant is:

    final <data type> <variable name> = <value>;

For example:

    final double PI = 3.14159265;

Say we want to write a program that will add two numbers and display the result. We need space to hold the two numbers and the result (three variables). If we want to add integers the variables will be of type short, int or long.

    // declare space for the three variables
    int number1, number2, sum;
    // give value of 0
    number1 = number2 = sum = 0;
    or
    // declare space for the three variables and give initial values of 0
    int number1 = 0, number2 = 0, sum = 0;

Primitive Data Types
Data types specify the type of data and indicate how much memory is required to hold the data. There are eight primitive data types in Java, six numeric types: byte, short, int, long, float, double; and char and boolean types.
Integers consist of a sign (+ or -) and a whole number value. Java has three integer data types: short, int, and long. Short integers are represented using 16 bits, thus short integers will be in the range \(-32768\) \((-2^{15})\) to \(+32767\) \((-2^{15}-1)\). Integers are represented in 32 bits and will be in the range \(-2^{31}\) to \((-2^{31}-1)\). Long integers are represented in 64 bits.

Some integer variable declarations:
```java
short num1, num2;
int num1 = 0, num2 = 7, num3 = -43;
long num = 2000000L;
```

Note: When assigning an integer literal to a long integer variable append a capital L on the end of the integer literal.

Real numbers are represented in floating point notation. Java has two floating point data types, float for single precision real numbers and double for double precision real numbers.

Some real number variable declarations
```java
float num;
double num1 = -7.567, num2 = 0.0;
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Memory in Bytes</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>4</td>
<td>An integer between (-2,147,483,648) and (2,147,483,647).</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
<td>An integer between (-32,768) to (32,767).</td>
</tr>
<tr>
<td>long</td>
<td>8</td>
<td>An integer between (-9,223,372,036,854,775,808\L) to (9,223,372,036,854,775,807\L).</td>
</tr>
<tr>
<td>byte</td>
<td>1</td>
<td>Uses 8 bits to represent a number from -128 to 127.</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
<td>Represents numbers with fractional parts such as 12.3456. Valid values span 6-7 decimal digits</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
<td>Valid values span 15 decimal digits.</td>
</tr>
<tr>
<td>char</td>
<td>2</td>
<td>Represents single characters between single quotes.</td>
</tr>
<tr>
<td>boolean</td>
<td>1</td>
<td>Can be either true or false.</td>
</tr>
</tbody>
</table>

The table above lists all of the Java primitive data types and a short description.

**Operators**
Java supports addition (+), subtraction and negation (-), division (/), multiplication (*), and modulus (%). The results of operations are assigned using the assignment operator =. The format for an arithmetic operation is:

```java
<object> = <expression>;
```

The object on the left hand side must be a simple object. The expression on the right hand side can consist of variables, literals, and arithmetic operators. It is evaluated using standard precedence and the result is assigned to the object on the left hand side. Any objects on the right hand side are unchanged when using basic operators. Note the = is not a comparison between the two items, it is setting the left hand side to the value of the right hand side.

+, -, and * all work like expected. Division of floats or doubles works as expected but integer division discards anything after the decimal. Modulus returns the remainder of a division.

Arithmetic operator precedence is same as you are used to.
- Parenthesis
- *, / and % from left to right.
- + and - from left to right.

To continue our addition example, once we have values for the two numbers we can generate the sum:
```java
// compute sum
```
Input and Output
We need a way to read in data from the user and to send the result to user. This can be done from the console, dialog windows, or GUIs.

Console Input and Output
Java has many predefined classes grouped into categories called packages. This is referred to as the Java class library or Java Application Programming Interface (Java API). To use the objects and methods of a class, you must import the package containing the class. The System class is in the `java.lang` package which is automatically imported. It contains classes for performing system operations like console input and output. `System.in` and `System.out` refer to standard input and output devices respectively. The methods `print(…)` and `println(…)` are part of `System.out` and display strings to the console. The Scanner class is in the `java.util` package and can be used to create objects that can read data from `System.in`.

Consider rewriting the addition program to read in data from the command line and display the sum in the command line:

```java
import java.util.Scanner;

public class AdditionConsole {
    public static void main(String[] args) {
        int number1, number2, sum = 0;

        // set up console input
        Scanner scanner = new Scanner(System.in);

        // read in numbers from user
        System.out.print("Enter number 1 (an integer): ");
        number1 = scanner.nextInt();
        System.out.print("Enter number 2 (an integer): ");
        number2 = scanner.nextInt();

        // compute sum
        sum = number1 + number2;
        System.out.println("Sum is: " + sum);

        // terminate the program
        System.exit(0);
    }
}
```

Note: the Scanner class also has methods `nextLong(…), nextDouble(…)`, etc to return the other Java data types.

GUI Input and Output
The `showInputDialog(…)` method of the `JOptionPane` class creates a dialog box with text, a place for the user to enter a string, and an OK and Cancel button. The `showInputDialog(…)` method is a static method in `JOptionPane` and is called using dot notation as discussed before. Some methods return values and can be used in assignment statements. The `showInputDialog(…)` method has four arguments, the third and fourth are optional. The first argument specifies the window the dialog box will appear on top of (the parent window; null means no window is specified), the second is the
prompting text to be displayed in the box, the third is the title of the dialog box, and the fourth indicates the type of dialog box.

The `showInputDialog(…)` method returns a string by default, but we need an integer. The string value can be converted into an integer using the static method `parseInt(…)` in the `Integer` class. Each Java primitive class has an associated wrapper class that provides object methods for the primitive data types. The wrapper classes are `Byte`, `Short`, `Long`, `Integer`, `Float`, `Double`, `Character`, and `Boolean`. Each of these classes contains a `parseClass(…)` method to convert certain data types to a primitive data type.

Taking values from a user through one of these dialog boxes and converting it into an integer is shown in the `Addition.java` file from earlier examples:

```java
import javax.swing.JOptionPane;

public class Addition {
    public static void main(String[] args) {
        String number;
        int number1, number2, sum = 0;

        // read in numbers from user as strings and convert to ints
        number = JOptionPane.showMessageDialog(null, "Enter first integer");
        number1 = Integer.parseInt(number);
        number = JOptionPane.showMessageDialog(null, "Enter second integer");
        number2 = Integer.parseInt(number);

        // compute sum
        sum = number1 + number2;

        // display the sum
        JOptionPane.showMessageDialog(null, "The sum is " + sum, "Results",
                                      JOptionPane.PLAIN_MESSAGE);

        // terminate the program
        System.exit(0);
    }
}
```

We examined GUI output with `showMessageDialog(…)` in detail in Lecture 2. Refer to those notes if you need to be refreshed on output dialog boxes.

You have to be careful when using the `+` operator when working with output. When trying to output numbers, you can inadvertently add numbers instead of displaying them in order. When used in conjunction with strings, the `+` symbol is the concatenation operator. This allows you to output the value of a variable within in an output statement.

The `exit` method of class `System` terminates the application. The argument 0 indicates that the program terminated successfully. By convention, a nonzero status code indicates abnormal termination. This statement is required in programs that use GUI elements such as dialog boxes.

*Class Exercise #6 – Addition.java*