Introduction to Programming with Java

What is Programming?

A computer program is a list of specific instructions that the computer can execute. We can tell the computer to print something, to add numbers, to repeat a sequence of instructions, etc. Just like there are rules to writing sentences in English, there are rules we must follow when writing computer programs. If we don't follow the rules, the computer won't understand what we want.

Hello, World Program

Here is a very simple Java program that prints out "Hello, World!" to the screen:

```
public class Hello
{
     public static void main(String[] args)
     {
          System.out.println("Hello, World!");
     }
}
```

This text is stored in the file `Hello.java`. We will learn much more about what everything means as the course progresses, but for now let's discuss each line separately:

1) **`public class Hello`** – this line begins a *class* in our program. For now, just remember that the name you give the class (in this case, "`Hello`") must match the name of the file (in this case, `Hello.java`). Capitalization matters – we could not name the class "`hello`".
2) **`{`** – this bracket opens up the class
3) **`public static void main(String[] args)`** – this line declares the main method for our program, which is where the program begins. For now, just copy this line into all your programs
4) **`{`** – this bracket opens up the main method
5) **`System.out.println("Hello, World!");`** – this line prints "Hello, World!" to the screen
6) **`}`** – this bracket ends the main method
7) **`}`** – this line ends the class

Compiling

Once you have written a computer program, you need to *compile* it. This process does two things:

1) Checks to see if your program has any errors
2) Converts your program into an executable file that the computer can run

To compile your program, open a **Command Prompt** (`Start->Run`, type `cmd`). Use

```
cd dirName
```

to change into a directory called `dirName`. You can also do:

```
dir
```

to list the contents in the current directory. Once you have changed to the directory where your program is saved, you can compile your program like this:

```
javac Name.java
```

(where `Name.java` is the name of your program). For example, to compile the Hello, World program you would do:

```
javac Hello.java
```

If your program has any errors, the compiler will print a list of what is wrong and where the problem is. If there are no errors, it will create the file `Name.class` (where `Name` is the name of the program file). For example when you compile the Hello, World program it will generate the file `Hello.class`.

If you get an error saying that the `javac` command is not recognized, please refer to the tutorial on Compiling and Running Java programs.

Executing
After you have compiled your program and generated a class file, then you are ready to run it. At the command prompt, type:

```
java Name
```

where `Name.class` is the name of your class file. To run the Hello, World program you would type:

```
java Hello
```

to run the program.

Interactive Development Environments (IDEs)
For this class, you can just write your programs in a text editor (like Notepad) and compile and run your programs from the command window (as shown above). However, you may find that you prefer an interactive development environment that combines these steps in one application. If you are interested, check out the tutorial on Compiling and Running Java programs.

Comments

It is handy to add *comments* to your program that explain what different parts are doing. A comment is text in your program that is not actually part of the program itself. A one-line comment begins with a `//`. For example:

```
//This is a one-line comment
```

A multi-line comment begins with a `/*` and ends with a `*/`. For example:

```
/*This comment spans
  multiple lines */
```

All comments are ignored by the compiler.

Brackets

We will be talking more about the structure of computer programs later on, but for now all your programs should have the following format:

```
public class Name
{
     public static void main(String[] args)
     {
          //statements, such as printing
     }
}
```

Again, this program should be stored in the file `Name.java`. The brackets in programs do not have to be arranged like the example above – there are many different formats that people use. For example, this is the format that I prefer:

```
public class Name {
     public static void main(String[] args) {
          //statements, such as printing
     }
}
```

Additionally, your program does not have to be spaced like the example above. The following program is also perfectly fine:

```
public class Name
{public static void main(String[] args){
//statements, such as printing
}}
```

However, this program is much more difficult to read. It is a good idea to always indent (with a tab) every time you open a bracket. When you close that bracket, do not tab over that line.

Variables

In your programs, you can declare *variables* that help you store information. These storage devices are called variables because you can *vary* the information stored there.

Types

In Java (unlike some other programming languages), when you declare a variable you must give it a *type*. The type specifies what kind of information you plan to store in the variable. Below is a table of common types in Java and their uses:

Type	Use
`int`	whole numbers, such as `4` and `-23`
`char`	single characters, such as `'a'`
`double`	numbers with decimals, such as `3.14`. These can also be referred to as floating point numbers.
`boolean`	boolean values: either `true` or `false`
`String`	a sequence of characters, such as `"Hello"`

There are other types in Java, but these are the most common.

Declarations

You can declare a variable like this:

```
type name;
```

Here, `type` is one of the types in the table above, and `name` is the name we're giving the variable. We must follow these rules when name a variable:

1) The name should be a sequence of upper-case letters, lower-case letters, numbers, and underscores
2) The first character in the name must be either a letter or an underscore

Here are some example of variable declarations:

```
int num;
double val1;
char _letter;
String Name;
boolean check_done;
```

Variables in Java are *case-sensitive*. This means that if you change the capitalization in a variable name, then it does NOT refer to the same variable. For example, we can do this:

```
int num;
int Num;
```

and num and Num will be two different variables. (Note: this practice is not recommended because it causes confusion.)

Assignments

After we have declared a variable, we can *assign* it a value. We CANNOT use a variable in any way before it has been declared. A variable assignment looks like this:

```
name = value;
```

Here, name is the name of the variable that you declared, and value is the value you're giving it. The value you assign a variable must have the same type as the variable. (For example, if the variable is an int, then you can't store a number like 2.34 in it.) Here are some examples of valid assignments using the variables declared above:

```
 num = 42;
val1 = 3.21;
_letter = 'A';
Name = "Fred";
check_done = true;
```

Notice that single characters (chars) must be encolosed in single quotes, while string must be enclosed in double quotes.

We can also declare a variable and assign it a value at the same time. For example, we can do:

```
double pi = 3.14159;
```

Type Casting

We cannot give a value to a variable if it does not have the appropriate type. For example, we cannot do:

```
//This is illegal!
int num = 2.3;
```

However, sometimes we want to convert a value to have the appropriate type. We can do this by putting the type we want in parentheses in front of the value. For example:

```
int num = (int) 2.3;
```

This statement converts 2.3 into an int, so that num now has the value 2. We can do a similar thing to convert the types of variables:

```
double x = 6.75;
int y = (int) x;
```

Now `y` has the value 6. This conversion of one type to another is called *type casting*.

Literals

A *literal* is a constant value of a particular type, written the way it would appear in a program. Here are some examples of literals of different types:

Type	Sample Literals
`int`	34, -17, 0
`double`	3.42, 12.0, -14.7
`char`	'A', 'a', '2', '!'
`boolean`	true, false
`String`	"apple", "14", "a"

Notice that character literals must be enclosed in single quotes, and string literals must be enclosed in double quotes.

Operations

Now that we can declare variables and assign them values, we want to be able to perform calculations with them.

Mathematical Operations

These operations deal with numerical data (ints and doubles). Here are the mathematical operators that you can use:

- **`+`**: addition
- **`-`**: subtraction
- **`*`**: multiplication
- **`/`**: division
- **`%`**: modulus (returns the remainder of dividing one whole number by another)

Here are some examples:

```
//gives num the value 6
int num = 2*3;

//gives x the value 3.3
double x = 6.6/2.0

//this is integer division, so we drop the decimal portion
//gives result the value 2
int result = 7/3;

//7/3 is 2 remainder 1
//gives mod the value 1
int mod = 7%3;
```

We can also use multiple mathematical operators at once, and we can involve variables in the expressions. Here are some more examples:

```
int x = 3;
double y = 4.4;
int z = 10;

//gives result1 the value 29
int result1 = x*z – z%x;

//gives result1 the value 26
result1 = (int)(z/y + 73/x);
```

Notice that these examples are NOT the same as equations in algebra. In an equation, the expression on the left-hand side equals the expression on the right-hand side. In these examples, we are assigning a variable (the left-hand side) to have the VALUE from the expression on the right-hand side.

We can also do something like this:

```
int num = 1;
num = num + 1;
```

This updates the value of `num` to be one bigger than the old value of `num`. Now `num` has the value 2.

We CANNNOT do something like this:

```
//Illegal!
num + 1 = num;
```

OR

```
//Illegal!
5 = num;
```

The left-hand side must be a single variable, and we are updating that variable's value to be the result of the right-hand side.

Expressions

An *expression* is a computation of variables and constant values (*literals*) using mathematical or other operators. When we evaluate an expression, we get a value as a result. For example, variable assignments might look like:

```
variable = expression;
```

For example:

```
int x = 7*(3-2)+1;
```

Here, **`7*(3-2)+1`** is an expression. We evaluate it first, and then the result is stored in the `x` variable.

Shortcuts

There are several simple operations that you will find yourself doing over and over, such as adding one to a variable or multiplying a variable by a single value. For example:

```
int num = 6;

//gives num the value 8
num += 2;
```

This statement adds 2 to `num`, so that `num` now has the value 8. It is equivalent to the statement:

```
num = num + 2;
```

We can also do something similar with `-=`, `*=`, `/=`, and `%=`. Here are some examples:

```
int x = 1;
//gives num the value 7
num -= x;

//gives num the value 35
num *= 5;

//gives num the value 11
num /= 3;

//gives num the value 1
num %= 2;
```

There are other shortcuts we can use if we want to add one to a variable or subtract one from a variable:

```
int val = 4;
//adds one to val, now val is 5
val++;

//subtracts one from val, now val is 4
val--;
```

The statement `val++` is equivalent to:

```
val += 1;
```

It is also equivalent to:

```
val = val + 1;
```

(A similar comparison is true for the statement `val--`.)

Conditional Operations

There is another group of operators we can apply to boolean variables (variables that can be either true or false). Here is a list of these *conditional* operators:

==: checks whether two values are equal
!=: checks whether two values are not equal
<: checks if the first value is less than the second value
>: checks if the first value is greater than the second value
<=: checks if the first value is less than or equal to the second value
>=: checks if the first value is greater than or equal to the second value
&&: checks if both variables or expressions are true
||: checks if at least one of two variables or expressions is true
!: switches a variable or expression from false to true, or from true to false

Here are some examples:

```
boolean a = true;
boolean b = false;
int x = 4;
int y = 3;
int z = 3;

//result is set to false (x does not equal y)
boolean result = (x == y);

//result is set to false (y does equal z)
result = (y != z);

//result is set to false (x is not less than z)
result = (x < z);

//result is set to false (z is less than x, but a does not equal b)
result = (z < x) && (a == b);

//result is set to true (z is less than x – it doesn't matter whether a equals b)
result = (z < x) || (a == b);
```

```
//result is set to true (!b is true, and a is true)
result = !b && a;
```

Printing

We have already seen how to print text, but this section will review printing and go into more detail about what you can do.

Basic

There are two commands for printing text:

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

The first command prints `Hello` to the screen, and then moves down to the next line. This way, if you print something else it will appear on the line below. The second command prints `Hello`, but stays on the same line. If we printed something else, it would appear on the same line as `Hello`.

Of course, we can substitute any string we like for `"Hello"`. The string we want to print must be in `""` quotes, but the quotes do not actually get printed.

Printing Variables

In addition to printing strings enclosed in `""`, we can also print the values of variables. To do this, we just put the variable name inside the `System.out.println` or `System.out.print` parentheses. When that line is executed, the VALUE of the variables gets printed to the screen.

Here are some examples:

```
int num = 4;
char c = 'A';
double val = 7;
boolean flag = true;
String s = "Cat";

//prints 4
System.out.print(num);

//prints A on the same line as 4
System.out.println(c);
```

Concatenating

We can output text that has both variable values and regular strings with a single `System.out.println` statement. This can be accomplished by using the **concatenation** (+) operator. Here's an example:

```
String name = "Fred";
int age = 20;
System.out.println(name + " is " + age + " years old");
```

This prints out "`Fred is 20 years old`". Notice that if you want to print the value of a variable, just list the variable (not in quotes). If you want to print normal text, put the text in quotes. To combine variables and text, separate them by a +. This plus concatenates the variables and text by pushing them together to form one big string.

User Input

We can get user input by printing a prompt to the command-line, and then having the user type the information right after the prompt.

Basic

Here's how to get user input:

1) Add the following to the top of your file:

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

2) Create a **`Scanner`** to help you read in the input. For now, just add this line ONCE, before any user input. (Don't do this again even if you want more input.)

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

3) Print out a descriptive prompt (use `System.out.print` so it will be on the same line as the input):

```
System.out.print("Enter name: ");
```

4) Read in the input with the command `s.nextLine()`. This command returns the `String` that was typed by the user.

```
String name = s.nextLine();
```

Here's the full program:

```
import java.util.*;
public class ConsoleInput {
    public static void main(String[] args) {
```

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

            System.out.print("Enter name: ");
            String name = s.nextLine();

            //now the name variable holds the name typed by the user
        }
    }
```

Conversions

We don't always want to read in strings from the user. We may also want to read in numbers. To do that, we need a way to convert from a string to a type like an int or double.

We cannot convert from strings to ints or doubles by using type casting. We will discuss this more later on, but strings are object variables while other types of variables have primitive types. For now, just remember that strings are treated differently.

There are special parsing methods available to turn a string into a double or int. Here's how to use them:

```
String str1 = "345";
String str2 = "3.76";
String str3 = "A";

//converts from a string to an int
int num1 = Integer.parseInt(str1);
//converts from a string to a double
double num2 = Double.parseDouble(str2);
```

Now that we know how to convert strings to different types, we can read in things like numbers and characters. Here is an example:

```
Scanner s = new Scanner(System.in);
System.out.print("Enter your age: ");
String str = s.nextLine();
int age = Integer.parseInt(str);
```

At the end of this example, the `age` variable holds the number typed by the user. We can also read in the value typed by the user and convert it to the appropriate type all in one line:

```
Scanner s = new Scaner(System.in);
System.out.print("Enter your age: ");
int age = Integer.parseInt(s.nextLine());
```

NOTE: Scanners in Java have techniques for reading in things like doubles and ints without using the conversion methods above. We will discuss these techniques later in the course.

Examples

We have now seen how to perform operations on variables, and how to get user input. Let's practice with two complete example programs.

Area of a Rectangle

Suppose we want to ask the user for the length and width of a rectangle, and then print out the area of the rectangle. Here are the steps we will need:

- Declare all the variables for the program. This includes the *length*, *width*, and *area* of the rectangle.
- Set up a `Scanner` since we will need user input.
- Ask the user to enter the width. Read what was typed into the *width* variable (use the parsing command).
- Ask the user to enter the length. Read what was typed into the *length* variable (use the parsing command).
- Assign *area* to be the *width* times the *length* (which is how the area of a rectangle is computed)
- Print the *area* to the screen

Here is the complete program:

```
import java.util.*;

public class Rectangle
{
     public static void main(String[] args)
     {
          Scanner s = new Scanner(System.in);
          int length;
          int width;
          int area;

          System.out.print("Enter the width: ");
          width = Integer.parseInt(s.nextLine());
          System.out.print("Enter the length: ");
          length = Integer.parseInt(s.nextLine());

          area = length*width;

          System.out.print("The area is: ");
          System.out.println(area);
     }
}
```

Temperature Calculator
Next, suppose we want to ask the user for a temperature in Celsius, and then print out the result in Fahrenheit. Here are the steps we will need:

- Declare all the variables for the program. This includes the temperature in Fahrenheit (*f*) and the temperature in Celsius (*c*).
- Set up a `Scanner` since we will need user input.
- Ask the user to enter the temperature in Celsius. Read what was typed into the *c* variable (use the parsing command).
- Assign *f* to be 9/5**c* + 32 (that is the formula for converting from Celsius to Fahrenheit)
- Print the temperature in Fahrenheit (*f*) to the screen

Here is the complete program:

```
import java.util.*;

public class Temperature
{
     public static void main(String[] args)
     {
          Scanner s = new Scanner(System.in);
          int f;
          int c;

          System.out.print("Temperature in Celsius: ");
          c = Integer.parseInt(s.nextLine());

          f = 9.0/5.0*c + 32;

          System.out.print("In Fahrenheit: ");
          System.out.println(f);
     }
}
```

Note that we changed the conversion formula to be 9.0/5.0*c + 32 instead of 9/5*c + 32. Can you think why that would be?

(Remember *integer division* – if we divide two whole numbers, the decimal portion of the result is cut off. Since 9 and 5 are both whole numbers, 9/5 is just 1, the whole number portion. However, 9.0 and 5.0 have decimals, so 9.0/5.0 is 1.8, as we want.)