COP 3330 – Object Oriented Programming in Java
Variables, Input, Assignment, and Arithmetic Expressions

Types in Java

Every type in Java is either a primitive or a reference type. The primitive types are those built into the language that are values. The primitive types in Java are:

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

Of these, we will typically use:

```
int, long, double, boolean, char
```

For integers, the type int has the same storage as the type int in C, able to store integers in the range $[-2^{31}, 2^{31} - 1]$. This what we'll typically use for integer data.

If we need to make intermediate calculations that exceed $2^{31} - 1$, then we'll have to use long, which can store integers in the range $[-2^{63}, 2^{63} - 1]$.

For floating point numbers, we'll just always use double. This has the same storage as double in C, allowing for about 12 digits of precision with a magnitude as large as 10^{308}/as small as 10^{-308}.

C does not have an explicit boolean type, but Java does. Thus, there's no "trickiness" dealing with integers being interpreted as true or false. If you put an integer in a construct in Java when it's expecting a Boolean expression, then the code won't compile. The two possible values a boolean variable in Java can have are `true` and `false`. Note that for Java, these literal values are all lowercase. (In Python they are `True` and `False`.)

Finally, individual characters can be stored in a char variable. Java uses underlying Ascii values just like both Python and C. Whereas you don't have to give variables types in Python before using them, in Java, you must declare a variable with a type, just like C.

Java is fairly strongly typed. This means that in expressions, you MUST have components such that their types match. If they don't, often times the compiler will give an error. (In C, typically these are warnings and in Python, the interpreter tries to figure out what type would make something work and then sometimes causes a run-time error.) While this sounds annoying, it's pretty helpful and tends to weed out some errors before you even run a program, which is nice.

The rules for defining variables in Java are the same as C, with one notable exception:

Variables in Java are required to be assigned to a value, otherwise the code won't compile the first time the variable is referenced in an expression.

However, if you declare an array of a primitive type, then Java will default assign each of the values in the array as follows (without you having to do so):

All integer types (including char) are defaulted to a value of 0. All floating point types are defaulted to a value of 0.0. All boolean variables are defaulted to false.

First Program with Variables

A classic program shown in introductory programming classes to illustrate the use of a variable is one that finds the area of a circle. Here we'll illustrate the use of the types int and double, the use of a constant (identifier that doesn't change) as well as printing out both a string literal and the value of a variable:

```
// Arup Guha
// 7/7/03
// Short Program to illustrate the calculation of a formula in java, using
// variables and arithmetic expressions.

public class Circle {

      final public static double PI = 3.14159265358979;

      public static void main(String[] args) {

            // Calculate area of circle.
            int radius = 30;
            double area = PI*radius*radius;

            // Print out area.
            System.out.print("The area of a circle with radius "+radius);
            System.out.println(" is "+area);
      }
}
```

Constants in Java

Python does not have constants and C uses either #define or the const keyword to define them. In Java, we should put them as class variables, before main, so they are accessible to all methods in the class. The keyword **`final`** indicates that the identifier defined is a constant and can't change value after it's assigned. Constants have to have a type in Java as well as an identifier name. Finally, a constant is never part of an object, so the keyword static needs to be used as well. There is a built in PI in the math class in Java, but we'll learn that later. Here we define our own constant PI. The conventions is for constant identifiers to be in all capital letters. We use constants in Java for the same reasons we use them in C: for readability and to make program maintenance easier.

Variable Declaration, Assignment Statement

To declare a variable, put the type first, followed by the variable name. To assign it to a value, use the equal sign for assignment and the expression you want to assign it to, to the right of the equal sign. The rules for variable declaration and assignment are the same as in the C language.

You can declare more than one variable of the same type on a single line (separated by commas). Variables are valid to use (scope) from the point they are declared until the close parenthesis within which they reside is hit.

We'll look at the rules for evaluating expressions later, but for now, they roughly work just like in C and Python, with typical precedence rules. If you don't know the precedence rules, you can always use extra parentheses to force your desired precedence.

The assignment statement works the same in Java as it does in C and Python. Step 1 is to evaluate the expression on the right, and step 2 is to change the value of the variable on the left to the value just computed in step 1.

Program Output

In your introductory class, you should have gone through a similar program step by step, looking at a picture of the status of boxes (showing the value of variables at each step). This works the exact same in Java as it does in other languages with primitive data types. The result of running this program is the following output:

```
The area of a circle with radius 30 is 2827.433388230811
```

For this class, I'll assume you already know how an assignment statement works and will draw the pictures as necessary, but not as frequently as I would in a student's first programming class. I will draw A LOT of pictures when we get to classes, because that will be new for all students with only Python or C background. (Python does have objects...but often times they aren't taught to introductory students.)

Reading Input

The easy way to read input is using a Scanner and reading token by token. Later in the semester we'll learn how to use a StringTokenizer with a Scanner to read input line by line as needed. But, since I like to keep input formats simple, most (if not all) of our assignments will be doable via reading input via tokens and Scanner only. One key piece of advice I have is that you never want to mix reading tokens with lines. If you want to read tokens, only read by token. If you want to read lines, only read lines.

Scanner class and Java Doc

The class that Java provides to help reading input is the Scanner class. To read input, you must create a Scanner object. Generally speaking, you will only ever create one Scanner object in a program. (But for other types of objects, this won't be true.) Note that to use a Scanner, we have to include the util packages as follows:

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

Thus, to use built in classes in Java, they have to be imported. To make life easier, we can import a whole package (set of classes) at once by using the star as shown above. If you only wanted to import Scanner, you would do:

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

since the Scanner class belongs in the util package.

Here is how to create a Scanner object that reads from the keyboard (standard input):

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

On the left hand side, we start with the type of the variable, which is Scanner, a built-in class in Java. We must give our variable a name, and I'll usually call my scanner from standard input stdin, to indicate that it's reading from standard input. (Every now and then I'll have examples where this variable name is different, just to illustrate that it can be.)

On the right hand side we have a call to the constructor. For all objects in Java, they must be "built" and the only way to build an object in Java (of any class/type) is by calling a constructor for it. To call a constructor, always use the keyword new followed by the name of the class of the object being built. Different constructors take in different parameters and the Scanner constructor takes a reference to where it will read from. A valid place to read from is System.in, which is the keyboard.

Now that the Scanner object is set up, we have several methods (functions) we can call on the Scanner object. To see what these are, pull up the Java doc for Scanner online. The easiest way to do this is just to Google, "Java Scanner documentation". Here is the link that it pulls up:

https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html

The Java doc lists common use as well as all methods that belong to a Scanner object. There are lots of these, but here is a listing of the ones we'll use most frequently to read by token:

Return type	Method name	Description
`int`	`nextInt()`	Scans the next token of the input as an int.
`double`	`nextDouble()`	Scans the next token of the input as a double
`String`	`next()`	Finds and returns the next complete token from this scanner.

The first is used to read in an integer, the second to read in a double and the last one to read in a String. They work a bit differently than C's reading apparatus. These are methods called on the Scanner object which return the next item read in. Thus, as is true of all functions that return something, it's best to call them within a larger line of code and not on a line by themselves. The most typical use in an introductory programming class would be to use a println statement to prompt the user to enter something, and then store what the user enters into a variable via an assignment statement. Here's a simple example from the next program:

```
int radius;
System.out.println("What is the radius of your pizza?");
radius = stdin.nextInt();
```

Here, after the print, assuming the user enters an integer, this will be read in from standard input (a token) and converted to an integer and returned. The value returned is then stored in the variable radius that was previously created.

First Program Reading Input: Circle2

Here is our first program where we read input from the user:

```
// Arup Guha
// 7/7/03
// Short program with input/output that allows user to choose the size of
// the radius of the circle. The program then outputs the area of this
// circle.

import java.util.*;

public class Circle2 {

      public static void main(String[] args) {

            // Declare necessary reading object.
            Scanner stdin = new Scanner(System.in);

            // Get size of radius from user.
            int radius;
            System.out.println("What is the radius of your pizza?");
            radius = stdin.nextInt();

            // Compute and print out area.
            double area = Math.PI*radius*radius;
            System.out.print("The area of your pizza with radius "+radius);
            System.out.println(" is "+area);
      }
}
```

We've already explained the first four lines. From there, we create a variable called area of type double and store in it the expression for the area of the circle as computed in the expression on the right, referencing the variable radius for which a value was just obtained. Then we print out our result. Notice that both prints use string concatenation with one string and one expression of type int or double. Java automatically converts the int expression to the equivalent string in the first print and converts the double expression in the second print to the equivalent string. Here is an example of the program running, user input in bold:

```
What is the radius of your pizza?
30
The area of your pizza with radius 30 is 2827.433388230811
```

Second Program Reading Input: Circle3

In an introductory programming class, one usually write several programs where you read some input from the user, do a simple set of calculations via assignment statement and print out the result. In general, these problems test problem solving skills and sequential execution. The more interesting ones require a bit more calculation by hand on paper, but after that result in relatively straightforward translation into code.

In this program, the user enters the length of fence they have, and we have to calculate the maximum area that can be enclosed by that fence. Mathematically, it can be proved that the best shape for the enclosure to maximize area is a circle. Thus, the problem boils down to taking in the input from the user that is equal to the circumference of a circle and then having to compute the area of that circle. We can use the fact that:

$$C = 2\pi r$$

and solve this equation for r: $r = \frac{C}{2\pi}$, so in our code, what we can do is read in the circumference, but then use it to calculate our radius and store that in a different variable. From there, we can make the same area calculation we previously made. Also, in this program, we are going to read in a double from the user, so we'll use the nextDouble() method.

Here is the code:

```
// Arup Guha
// 7/6/04: 2004 BHCSI Introduction to Java Course
// Calculates max area of enclosure given length of perimeter.

import java.util.*;
public class Circle3 {

	public static void main(String[] args) {

		// Declare necessary reading object.
		Scanner my_scanner = new Scanner(System.in);

		// Get length of the fence from user.
		double circum;
		System.out.println("How much fence have you bought in feet?");
		circum = my_scanner.nextDouble();

		// Compute the radius and area, then print out the area.
		double radius = circum/(2*Math.PI);
		double area = Math.PI*radius*radius;
		System.out.print("The area of your enclosure with "+circum);
		System.out.println(" feet of fence is "+area+" square feet.");
	}
}
```

Notice the use of parentheses on the right hand side of the radius calculation. This time, instead of using the constant PI that we made up, we use the one from the math library, also in the util package.

Printing to a fixed number of decimals: Circle4

There are multiple methods to output a floating point number to a fixed number of decimals places (rounded), but the easiest of these is simply to use the **`printf`** function. It works virtually the same in Java as it does in C. Here is the program in its entirety for convenience:

```
// Arup Guha
// 7/6/04: 2004 BHCSI Introduction to Java Course
// Edit of Circle3.java that shows how to output floating point
// expressions to a fixed number of decimals.

import java.util.*;

public class Circle4 {

      public static void main(String[] args) {

            // Declare necessary reading object.
            Scanner my_scanner = new Scanner(System.in);

            // Get length of the fence from user.
            double circum;
            System.out.println("How much fence have you bought in feet?");
            circum = my_scanner.nextDouble();

            // Compute the radius and area, then print out the area.
            double radius = circum/(2*Math.PI);
            double area = Math.PI*radius*radius;

            // Here is the area outputted rounded to two decimal places.
            System.out.printf("The area of your enclosure with %.2f feet of
fence is %.2f square feet.\n", circum, area);
      }
}
```

One difference is that in Java, the percent code for both a float and double is %f. Otherwise, the control string and placement of the parameters works exactly as the C programming language.

Arithmetic Expressions in Java

Precedence rules
The precedence rules in Java are similar to C. Here is an incomplete list of precedence:

1) Parentheses
2) Unary + or –
3) multiplication, division, mod
4) addition, subtraction, string concatenation
5) assignment statement

by default, just use parentheses if you aren't sure about the default precedence order. Too many parentheses will always work whereas not enough might not do what you want. (Balance with readability of course.)

Integer division and mod in Java
Integer division and mod work the same in Java as they do in C, but different than Python. (In Python the programmer explicitly denotes integer division as //.) In both Java and C, the compiler decides if a division is a floating point operation or an integer operation as follows:

If both operands are integer types, then an integer division is done.
If either operand is a floating point type, then a real number division is done.

For both Java and C, integer division removes the decimal part of the answer. Here are the values of several expressions using integer division:

200/201 evaluates to 0
14/3 evaluates to 4
-8/3 evaluates to -2

The mod operator (%) denotes the leftover when an integer division is done. Mod works a bit strange with negative numbers in Java (same as C but different than Python), but my advice is to just make sure the numbers you are working with are non-negative. You can always add multiples of the mod number to an expression to make sure it's non-negative before modding. Here are the values of several expressions using mod:

200%201 evaluates to 200
14%3 evaluates to 2.
63%7 evaluates to 0.
12345%1000 evaluates to 345.

mod is very, very useful. If you haven't seen too many examples of mod being used, please reference the following lecture from my C notes:

https://www.cs.ucf.edu/~dmarino/ucf/cop3223/lectures/ModNotes.pdf

Data Conversion

Although Java is strongly typed, some differences in type are allowed in statements/expressions. If we have a variable dollars of type int and a variable money of type double, and both have assigned values, then the statement

```
money = dollars;
```

is allowed even though types don't match. The computer sees that we are trying to store an integer in a double, which it allows by automatically converting the integer to an equivalent double (casting is how to do this explicitly).

But the following is NOT allowed:

```
dollars = money;
```

The reason for this is that this is considered a "narrowing" conversion from double to int. Most doubles can't be represented as equivalent ints. The "widening" conversion from int to double was allowed because ints can be reasonably stored as doubles. (There is a slight loss of precision sometimes but usually this is very slight.)

So, more generally, Java will allow assignments where a widening conversion needs to be made automatically, but not a narrowing one.

Casting

To force the narrowing conversion where data may be lost, a cast is needed. The following would be valid:

```
dollars = (int)money;
```

Here we are casting the expression on the right to an integer, which would just remove the decimal part of the number and return the expression as an integer so that it could be stored in dollars.

In general, we do a cast by putting the type we would like the expression we would like to change type to the left of the expression. Casting can be used to force a double division. Here is an example where both sum and numScores are integer variables with values stored:

```
double avg = (double)sum/numScores;
```

Overflow/Underflow

This is an issue in C as well. If an intermediate calculation exceeds the storage of a type, then the overlfow bits aren't stored, causing an overflow error. The main issue here is that values are not accurately preserved. Try this in a main method:

```
System.out.println(100000*100000);
```

It produces the output:

1410065408

This looks strange, but it's just 10000000000%2147483648. The latter value is 2^{31}. All the overflow bits are divisible by 2^{31}, since the highest positive bit stored has value 2^{30} in an integer.

Underflow refers to values that are "too small in magnitude" to store.

Arithmetic Promotion

In an expression of the form a + b, where a and b are different types (say a is of type long and b is of type int), then temporarily, Java produces a widening conversion for the computation of the expression but it can not ever change the type of a or b permanently.

These are most of the relevant rules. We'll see then in action as we need.