**CIS 3145 Week 4 Class Notes: Primitive Data Types**

**Chapter 7: Data and containers**

**Objectives**

* Understand that variables and constants are containers
* Know how to parse input, format output, and convert data types
* Use variables for calculations in programs
* Understand the role of order of operations
* Use Casting to convert from less to more precise data types.
* Use the Double and Integer classes to hold numbers as objects
* Use the NumberFormat class to format output
* Use the BigDecimal class for precise calculations

**Containers in Object Oriented programming**

**Variables** & **constants** hold data in one of eight **primitive data types** *or* in **Objects**. **Primitive variables** hold the value of the variable in memory, while **object variables** hold the memory location of the object in memory.

The primitive data types use one of four **categories** (Integer, Decimal, character & Boolean)

Java has

* four Integer primitive data types: **byte**, **short**, **int**, and **long**
* two Decimal primitive data types: **float** and **double**
* one-character primitive data type: **char**
* one Boolean primitive data type: **boolean**

Each variable or constant must be both **declared** and **assigned** a value

A **constant** is the same as a variable but its declaration has the key work “**final**” at the beginning

***Naming Conventions*** are used to make it easier to read the code. Avoid using abbreviations and capitalize the first letter of each ‘word’ in the **variable** name, except for the very first letter in the variable name: “myIntegerVariableName”. Use **ALL\_CAPS** and **underscores** for naming **constants**.

DON’T forget that Java is **case sensitive**!!!

**Casting**

The size of the Primitive Data types are

* byte, boolean 🡪 Bytes Size 1
* short, char 🡪 Bytes Size 2
* int, float 🡪 Bytes Size 4
* long, double 🡪 Bytes Size 8

Casting is the process of moving data from one data type to another. Because variables are not the same size we need to do either **Implicit** or **Explicit** casting.

When we convert from a smaller to larger variable this is called an ***Implicit*** casting (or widening conversion). This is easy because we go from a smaller to a larger sized container. The scale is obvious except for the when we implicitly cast a *long integer* variable (8 bytes) into a *float decimal* type variable (4 bytes). Floating point variables are more precise so the ‘bigger’ long variable can actually fit inside a ‘smaller’ float variable.

The problem is when we go from a larger to a smaller variable. This is called ***Explicit*** casting (or narrowing conversion). The explicit casting requires that we put the smaller data type in parenthesis before the larger variable. Details about the larger variable are lost. The smaller variable is less precise.

float f = 12.5;

is explicitly casted into a smaller variable…

int i = (int) f;

When this happens i will equal 12. The 0.5 is lost!

**Operators (Arithmetic and Assignment)**

The equal sign, ‘=’, does NOT mean equal!!!

The equal sign is an ASSIGNMENT symbol. The symbol means “take what is on the *right* side of the symbol and put it INTO the variable on the *left* hand side”.

int x = 5; 🡪 means put 5 into the variable x. Or **assign** x the value of 5.

The compiler will finish calculating any expressions on the right hand side before doing the assignment.

Arithmetic operations:

Binary operators: Addition (+), subtraction (-), multiplication (\*), division (/), Modulus (%)

Unary operators: Increment (++), Decrement (--), Positive Sign (+), Negative Sign (-)

There are two kinds of **symbols** used in programming arithmetic: **Binary** and **Unary**. The **Binary** works on two operands (values) and the **Unary** works on just one operand (value).

We are used to the **Binary** types of +, -, \*, and /: for example 2 + 2.

The % symbol is called modulus but is just the remainder: 7 % 2 is the value of 1. Seven divided by two is three with a remainder of one.

The Unary applies to a single operand (value).

The **shortcut** assignment operators are a strange **hybrid** of the assignment symbol and the arithmetic symbols.

The following statement uses an assignment operator shortcut symbol:

count += 1;

is the same as

count = count + 1;

It saves us the trouble of typing the variable ‘count’ twice.

Order of precedence of arithmetic expressions

Parentheses are the best way to dictate the order in which operations are performed. Any operations inside a set of parenthesis are done before operations outside of them.

The official order of operations is to first do:

* increments and decrements, then
* the positive and negative signs, followed by the
* multiplication, division, and remainder, and last
* addition and subtraction

If there is more than one operation of the same level next to each other they are done left to right, just like reading them.

**Miscellaneous classes**

This chapter also introduces classes used to manipulate data.

1. The Math class

This class has **static** methods so we don’t have to create an object to use the methods.

The math class does all sorts of operations like rounding, random number, absolute values, averages, medians, min, max, sin, and cos.

The following three variables can be used when finding the maximum value.

int x = 10;

int y = 25;

int maxValue = 0;

1. Using the Math class **max** method.

maxValue = Math.max(x,y);

**versus**

1. Using if statements

if (x < y)

maxValue = y;

else

maxValue = x;

Once a maxValue variable is defined it can be used as one of the two values in the max method. Thus we can repeatedly compare the **current maximum** value with a **new input** value. Whichever is larger gets assigned to the maximum value variable.

See the following for more details about the Math class: <http://docs.oracle.com/javase/8/docs/api/java/lang/Math.html>

1. BigDecimal Class

This class is a better representation of decimal values than what is done with the floating point primitive data types. The BigDecimal class is good for currency calculations.

The BigDecimal constructor is used to create a BigDecimal object. We can use int, double, long, or string variables as the input arguments for creating the objects.

This class has its own methods for doing arithmetic. Do not use the symbols used with primitive variables (+, -, \*, /) for BigDecimal variables, instead use the BigDecimal methods.

double tipVariable = 4.50;

BigDecimal dinnerTotalVariable.add(tipVariable); 🡨 gives the total charge