JAVA PROGRAMMING 1

Summer 2018 - Christian Hur

Unit 2 Lecture - Using Data

Readings: Chapter 2

## Data Types

A data type tells the type of data and how much memory it occupies a memory location in RAM. First, let’s look at a special group of data type known as **constants.** Constants are also called literal constants (i.e. numeric constants, string constants, character constants) or unnamed constants. Literal constants have no names (identifiers) associate with them and, therefore, they can only be used once. The following are examples of “literal constants”.

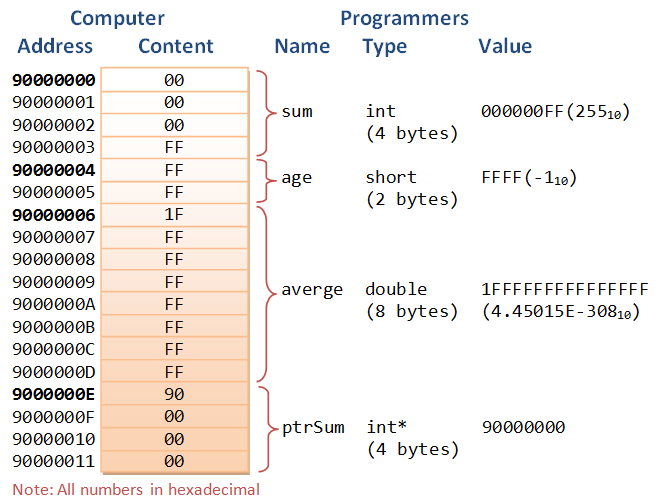
int age = **45**; //45 is a numeric constant

String name = “**Christian**”; //Christian is a string literal or string constant

boolean exist = **false**; //false is a keyword but also an unnamed constant

These constants can be replaced with **variables** so they can be reused numerous times in your code. A variable represents a memory location in the computer and is used to store a value such, as a literal constant or another variable. The name of a variable is known as an **identifier**.

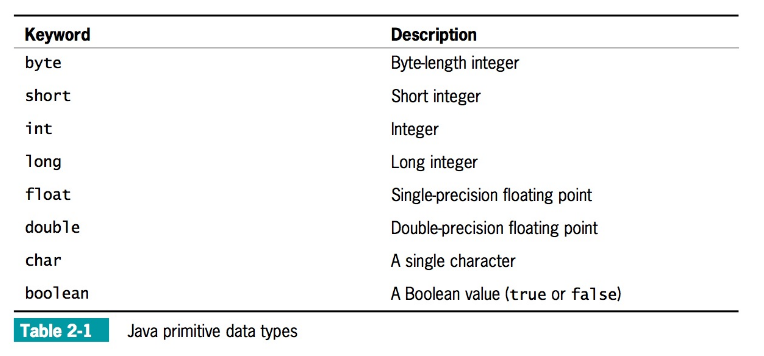
In the above example, the variables (or identifiers) are **age**, **name**, and **exist**, and each is of integer, string, and boolean data type, respectively. The pictorial below explains how variables (identifiers) and their values are represented and stored in the computer memory (RAM).



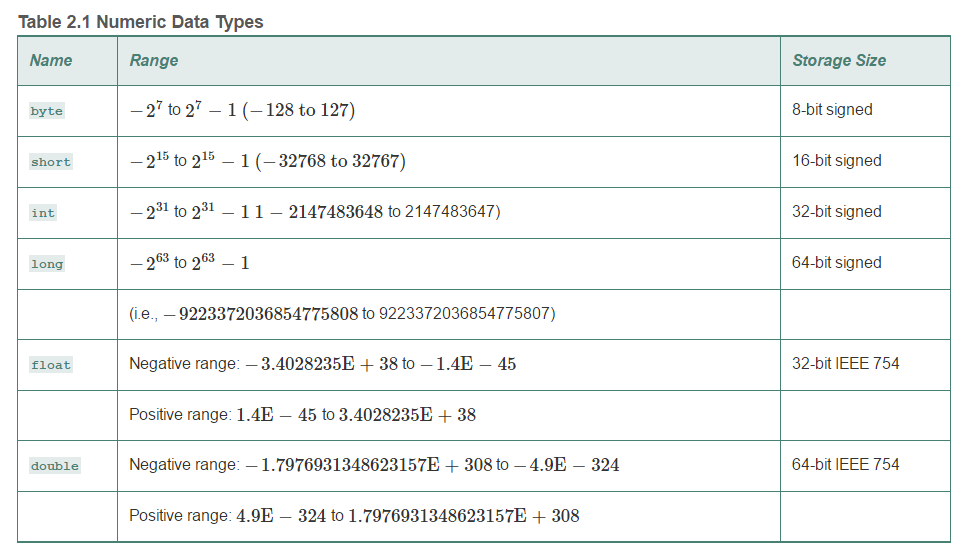
*(Source: NTU.edu)*

## Primitive and Reference Data Types

There are two major types of data: primitive and reference. A primitive data type also often referred to as simple data type and fundamental data type. Java provides eight primitive data types: 6 numeric, 1 character, 1 boolean.



Below is another table from a different text showing the range of data and their storage size.



## Declaring Variables

Before a variable can be used, it must be declared. Declaring a variable is a statement that allocates a memory location to store data. A variable is declared as follows:

1. A data type that identifies the type of data that the variable will store
2. An identifier that is the variable’s name
3. An optional assignment operator and assigned value, if you want a variable to contain an initial value
4. An ending semicolon

The following are examples of how variables can be declared.

int myAge;

double salary;

String topic;

boolean found;

Notice that there are not values assigned to the variables but only their data type. Once variables are declared, values can then be assigned to them. The following is an example of initialization which is the process of assigning initial values to variables.

myAge = 24;

salary = 50000;

topic = “Java Programming”;

found = false;

Instead of separating declaration and initialization statements as above, you can combine them together in one go as follows:

int myAge = 24;

double salary = 50000.45;

String topic = “Java Programming”;

boolean found = false;

You can even do this in a single statement by separating each identifier with a comma (,):

int myAge=24, double salary=50000, String topic=“Java 1”, boolean found=false;

However, this is not recommended if the list gets too long because it’s much arder to read.

If you have multiple variables of the same data type, you can declare them in a single statement as follows:

int myAge, myId, myFavoriteNumber; //no initialization

int myAge=24, myId=12345, myFavoriteNumber=777; //initialization

int myAge=24, myId, myFavoriteNumber=777; //initialization

## Variable Holds One Value at a Time (see page 61)

Each variable can hold just one value at a time. Suppose you have two variables, x and y, and x holds 2 and y holds 10. Suppose further that you want to switch their values so that x holds 10 and y holds 2. You cannot simply make an assignment such as x = y because then both

variables will hold 10, and the 2 will be lost. Similarly, if you make the assignment y = x, then both variables will hold 2, and the 10 will be lost. The solution is to declare and use a third variable, as in the following sequence of events:

int x = 2, y = 10, z;

z = x;

x = y;

y = z;

In this example, the third variable, z, is used as a temporary holding spot for one of the original values. The variable z is assigned the value of x, so z becomes 2. Then the value of y, 10, is assigned to x. Finally, the 2 held in z is assigned to y. The extra variable is used because as soon as you assign a value to a variable, any value that was previously in the memory location is gone.

## Declaring Named Constants

First of all, don’t confuse the terms “named constants” with “literal constants”. A named constant, also called a symbolic constant, is a variable (identifier) that is used to store a fixed value (of certain data type) that will remain constant throughout the program. The value assigned to a named constant cannot be changed at runtime, that is when the program is running.

A named constant differs from a variable in several ways:

* In its declaration statement, the data type of a named constant is preceded by the keyword **final**.
* A named constant can be assigned a value only once, and then it cannot be changed later in the program. Usually you initialize a named constant when you declare it; if you do not initialize the constant at declaration, it is known as a **blank final**, and you can assign a value later. Either way, you must assign a value to a constant before it is used.
* Although it is not a requirement, named constants conventionally are given identifiers using all uppercase letters, using underscores as needed to separate words.



For example, each of the following defines a conventionally named constant:

final int NUMBER\_OF\_DEPTS = 20;

final double PI = 3.14159;

final double TAX\_RATE = 0.015;

final string COMPANY = "ABC Manufacturing";

Named constants can be retrieved or used in an operation.

double payAmount = hoursWorked \* STD\_PAY\_RATE – numDependents \* DEDUCTION;

Once assigned a value, their value cannot be changed. The following is illegal.

final double PI = 3.14259;

.

.

PI = 3.14; //Illegal because PI has already been assigned

# Operators

Java uses the following operators to perform many types of operations. There are two special operators that require special attention.

The **assignment operator,** “=” (equal sign), is used to assign a value to a variable. It is not used to equate two values. It always requires two operands. The left operand is the variable receiving the value, the right operand is the value. Some examples:

a = 12;

b = 5;

sum = a + b;

total = sum \* getLarger(a, b);

The other special operator is the **concatenator**, “+” (plus sign). This can be confused with the mathematical addition operator, which is also a “+”. When used between two numeric variables, it acts as an addition. In the previous examples:

sum = a + b; //This is an addition.

However, when one of the operand is a string, it is used as a concatenator, which is used to join two or more strings together.

String str1 = “Hello”;

String str2 = “World”;

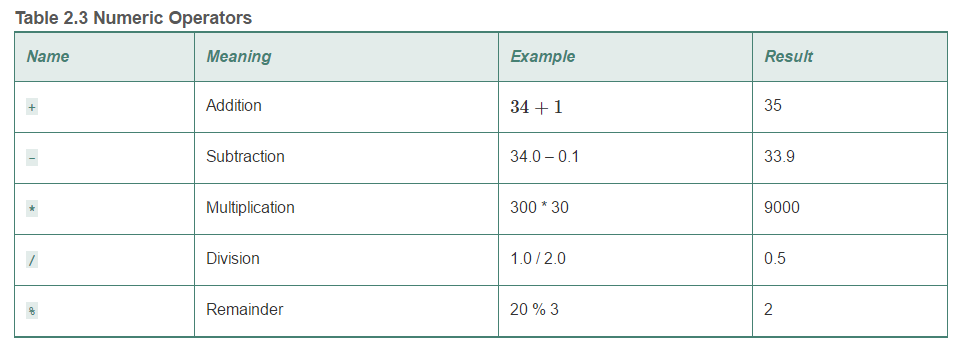
String str3 = str1 + str2; //string concatentor → “HelloWorld”

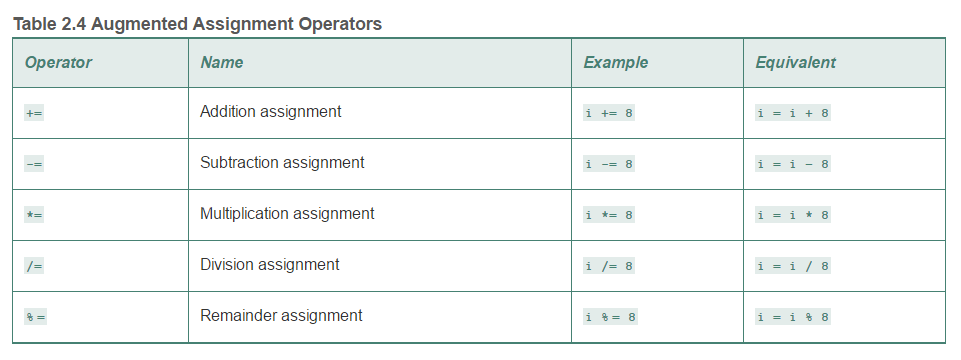
It can also be used as follow:

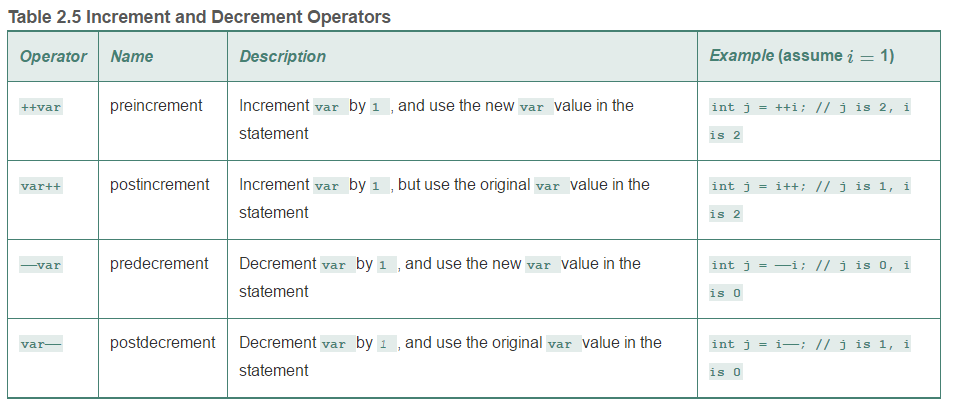
System.out.print(“The program says “ + str1 + “ “ + str2 + “!”);

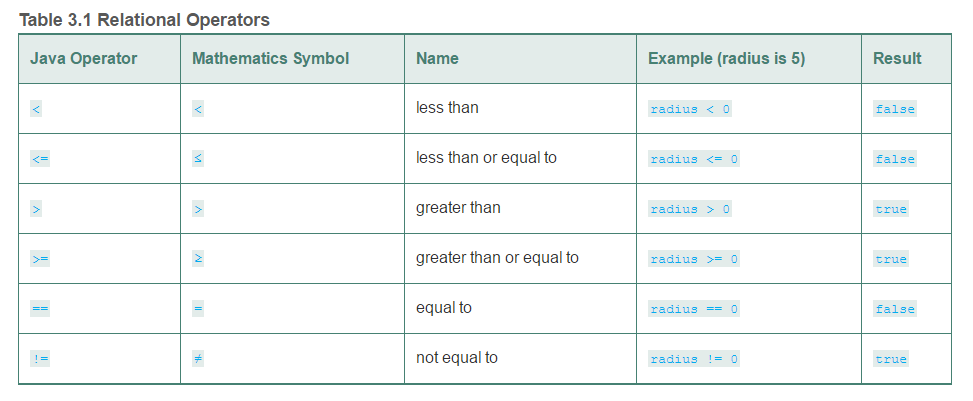
The output would read: The program says Hello World!

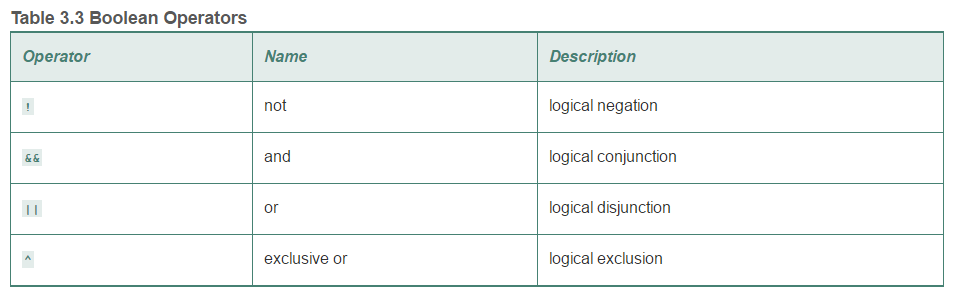
Below are many of the differen operators and their examples in Java.











# Scanner Class

You’ve learned how to write and display data to the screen by using the following statement.

String str = “Hello World”);

System.out.println(“I say “ + str);

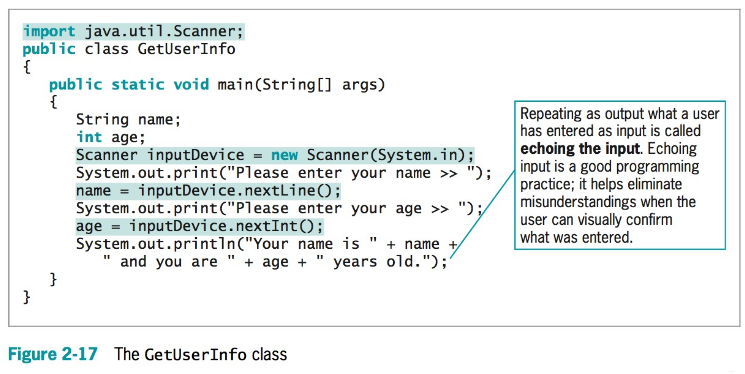
When you want to read input from the user via the standard input (from keyboard), you can use System.in, which refers to the standard input device (normally the keyboard). To read an input, you need to import a special class called Scanner.

import java.util.Scanner;

This is a class, which means you’d need to instantiate an object to use it. Declare a Scanner object as follow:

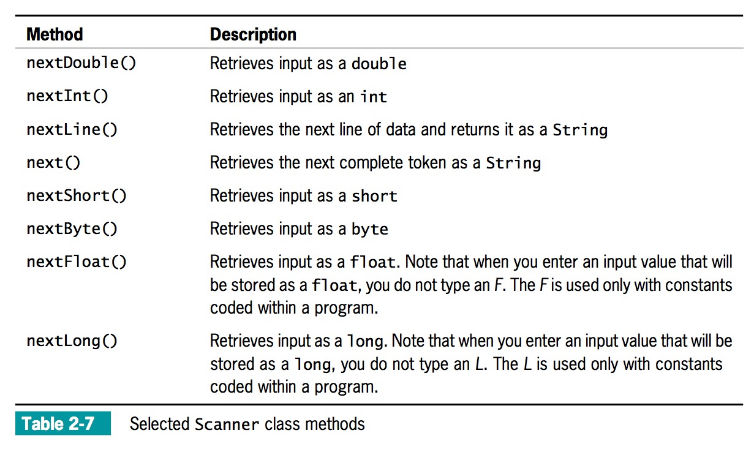
Scanner input = new Scanner(System.in);

Now, you can start using the object “input” to read input from the keyboard as in the following example.



* The first shaded statement imports the package necessary to use the Scanner class.
* The second shaded statement declares a Scanner object named inputDevice.
* The third shaded statement uses the nextLine() method to retrieve a line of text from the keyboard and store it in the name variable.
* The last shaded statement uses the nextInt() method to retrieve an integer from the keyboard and store it in the age variable.

Below is a list of some common Scanner methods you can use to read input.



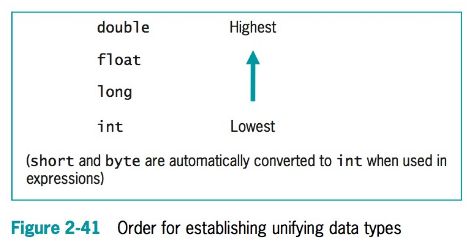
***Note****: All data read in from the keyboard are ALWAYS come in the form of char or String data type. This means that you will require to cast (or convert) the input to a specific data type in order to use it in any math operation. See page 101 for more information.*

## Type Conversion

When you perform arithmetic with variables or constants of the same type, the result of the operation retains the same type. For example, when you divide two ints, the result is an int, and when you subtract two doubles, the result is a double. Often, however, you might want to perform mathematical operations on operands with unlike types. The process of converting one data type to another is type conversion. Java performs some conversions for you automatically or implicitly, but other conversions must be requested explicitly by the programmer.

**Implicit Type Conversions**

When you perform arithmetic operations with operands of unlike types, Java chooses a unifying type for the result. Java performs an implicit conversion; that is, it automatically converts nonconforming operands to the unifying type.

****

When two unlike types are used in an expression, the unifying type is the one that is higher in the list in Figure 2-41. In other words, when an operand that is a type lower on the list is combined with a type that is higher, the lower-type operand is converted to the higher one. For example, the addition of a double and an int results in a double, and the subtraction of a long from a float results in a float.

For example, assume that an int, hoursWorked, and a double, payRate, are defined and then multiplied as follows:

int hoursWorked = 37;

double payRate = 16.73;

double grossPay = hoursWorked \* payRate;

The following code will not compile because hoursWorked times payRate is a double, and Java does not allow the loss of precision that occurs if you try to store the calculated double result in an int.

int hoursWorked = 37;

double payRate = 16.73;

int grossPay = hoursWorked \* payRate; // Failed because **int** is too small

// to hold a **double**.

**Explicit Type Conversions**

You can purposely override the unifying type imposed by Java by performing a type cast. Type casting forces a value of one data type to be used as a value of another type. To perform a type cast, you use a cast operator, which is created by placing the desired result type in parentheses. Using a cast operator is an explicit conversion. The cast operator is followed by the variable or constant to be cast. For example, a type cast is performed in the following code:

double bankBalance = 189.66;

// weeklyBudget is 47.415, 1/4 of bankBalance

float weeklyBudget = (float) (bankBalance / 4);

In this example, the double value bankBalance is divided by the integer 4, and the result is a double. Then, the double result is converted to a float before it is stored in weeklyBudget.

Without the conversion, the statement that assigns the result to weeklyBudget would not compile. Similarly, a cast from a float to an int occurs in this code segment:

// dollars is 47, the integer part of myMoney

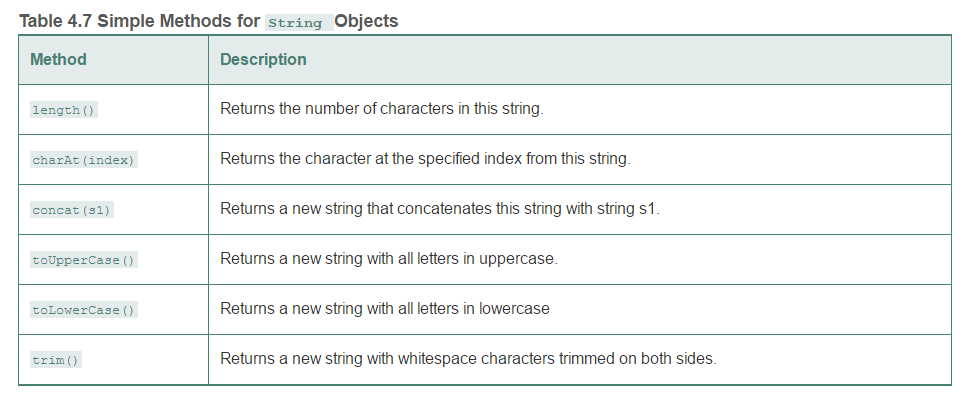
float myMoney = 47.82f;

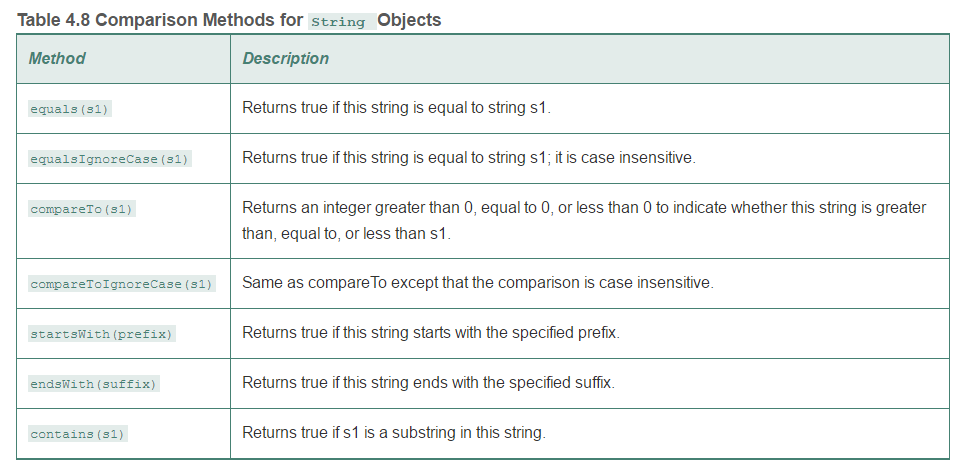
int dollars = (int) myMoney;

In this example, the float value myMoney is converted to an int before it is stored in the integer variable named dollars. When the float value is converted to an int, the decimal place values are lost. The cast operator does not permanently alter any variable’s data type; the alteration is only for the duration of the current operation. In other words, if myMoney was used again in the previous example, it would still be a float and its value would still be 47.82.

## Reference Date Types

* String type (String) - String is a predefined class in the Java library, just like the classes System and Scanner .
* Arrays
* Objects





if (string1 == string2)  
 System.out.println(*"string1 and string2 are the same object"*);  
else  
 System.out.println(*"string1 and string2 are different objects"*);

The == operator checks only whether string1 and string2 refer to the same object; it does not tell you whether they have the same contents. Instead, you should use the equals method.

if (string1.equals(string2))  
 System.out.println(*"string1 and string2 have the same contents"*);  
else  
 System.out.println(*"string1 and string2 are not equal"*);

## Numeric Type Conversions (Type Casting or Casting)

Casting is an operation that converts a value of one data type into a value of another data type.

**Widening a type** - casting a type with a small range to a type with a larger range

**Narrowing a type** - casting a type with a larger range to a smaller range

System.out.print( (double)1/2));

System.out.print( (int)4.5);