University of Kelaniya

Kesavan Selvarajah

BSc (UOK), MSc (Reading, UCSC)

Temporary Lecturer

Software Engineering Department

Faculty of Computing and Technology

skesa231@kln.ac.lk

Object-Oriented Programming

CSCI 21052 / ETEC 21062

Variables

- In Java, we can perform mathematical operations on variables.
- To do this, we first need to declare the type of data that will be computed in the operations.
- We can then assign values to the variables.
- Once we have assigned values to the variables, we can compute their sum and difference.
- To declare a variable, we use the following syntax:
  - **type variableName;**
- Where **type** is the data type of the variable, and **variableName** is the name of the variable.

Variables

- Suppose we want to compute the sum and difference of two numbers. Let's call the two numbers x and y.
- We must first declare x and y.
  - `int x, y;`
- When this declaration is made, memory locations to store data values for x and y are allocated.
- These memory locations are called **variables**.
- x and y are names or **identifiers** of these variables.
- Every variable has three properties:
  - A **memory location** to store the value
  - The **type of data** stored in the memory location
  - The **name** used to refer to the memory location.

Variables

- we can declare x and y separately as,
  - `int x;`
  - `int y;`
- However, we cannot declare the same variable more than once;
  - `int x,y;`
  - `int y;` **(this declaration is invalid)**
- There are six numerical data types in Java:
  - **Byte, Short, Int, Long -** are for integers
  - **Float, Double -** are for real numbers

Java numerical data types and their precisions

Data Type	Content	Default Value†	Minimum Value	Maximum Value
`byte`	Integer	0	−128	127
`short`	Integer	0	−32768	32767
`int`	Integer	0	−2147483648	2147483647
`long`	Integer	0	−9223372036854775808	9223372036854775807
`float`	Real	0.0	−3.40282347E+38‡	3.40282347E+38
`double`	Real	0.0	−1.79769313486231570E+308	1.79769313486231570E+308

Variables

- Here is an example of declaring variables of different data types:
  - `int i, j, k;`
  - `float numberOne, numberTwo;`
  - `long bigInteger;`
  - `double bigNumber;`
- After a variable is declared, we can initialize it by assigning some value to it. For example,
  - `int count;`
  - `count = 10;`
- At the time a variable is declared, it also can be initialized. For example,
  - `int count = 10;`

Variables

State of Memory

(A)
```
int firstNumber, secondNumber;
```

```
firstNumber  =234;
secondNumber = 87;
```

after (A) is executed

firstNumber

secondNumber

The variables **firstNumber** and **secondNumber** are declared and set in memory.

```
int firstNumber, secondNumber;
```

(B)
```
firstNumber  =234;
secondNumber = 87;
```

after (B) is executed

firstNumber 234

secondNumber 87

Values are assigned to the variables **firstNumber** and **secondNumber**.

Variables

Numerical Data

```
int number;
number = 237;
number = 35;
```

number

Object

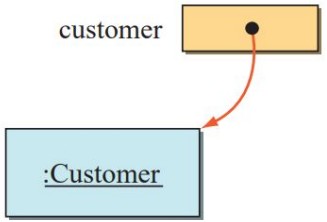
```
Customer customer;
customer = new Customer();
customer = new Customer();
```

customer

```
int number;
number = 237;
number = 35;
```

number 237

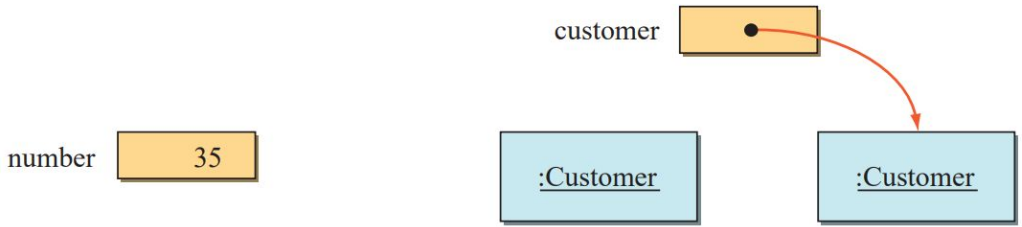
```
Customer customer;
customer = new Customer();
customer = new Customer();
```



Variables

```
int number;
number = 237;
number = 35;
```

```
Customer customer;
customer = new Customer();
customer = new Customer();
```



Variables

- The difference between object declaration and numerical data declaration:
  - For numbers, the amount of memory space required is fixed. The values for data type int require 4 bytes, and this won't change.
  - However, with objects, the amount of memory space required is not constant.
  - Declaring an object only allocates the variable whose content will be an address.
  - We use the new command to actually create an object.

Variables

- The difference between object declaration and numerical data declaration:
  - We don’t “create” an integer because the space to store the value is already allocated at the time the integer variable is declared.
  - Objects are called **Reference data types** because they contain addresses to the memory locations where the objects are stored.
  - In contrast to reference data types, numerical data types are called **Primitive data types**.

Variables

Data type	Description
Primitive	A data type that is predefined by the programming language. The size and type of variable values are specified, and it has no additional methods.
Reference	A data type that is not actually defined by the programming language but is created by the programmer. They are also called “reference variables” or “object references” since they reference a memory location which stores the data.

Variables

- In addition to the six **numeric data types** (Int, Long, Short, Byte, Float, and Double), there are two **non numeric primitive data types**.
  - The data type **Boolean** is used to represent two logical values true and false.

```
boolean raining;
raining = true;
```

  - The data type **Char** is used to represent a single character (letter, digit, punctuation marks, and others).

```
char letter;
letter = 'A';
```

Quick question

1. Why are the following declarations all invalid?

```
int a, b, a;

float x, int;

float w, int x;

bigNumber double;
```

Arithmetic Expressions

- An expression involving numerical values such as 23 + 45 is called an **arithmetic expression.**
- An arithmetic operator, such as **+** in the example, designates numerical computation.
- Division between two integers is called **Integer division**.
- The modulo operator % returns the remainder of a division.

Division Operation	Result
23 / 5	4
23 / 5.0	4.6
25.0 / 5.0	5.0

Modulo Operation	Result
23 % 5	3
23 % 25	23
16 % 2	0

Arithmetic Expressions

Operation	Java Operator	Example	Value (x = 10, y = 7, z = 2.5)
Addition	+	x + y	17
Subtraction	-	x - y	3
Multiplication	*	x * y	70
Division	/	x / y	1
		x / z	4.0
Modulo division (remainder)	%	x % y	3

Arithmetic Expressions

- An **Operand** in arithmetic expressions can be a constant, a variable, a method call, or other arithmetic expression, possibly surrounded by parentheses.
- The **addition** operator is called a **binary operator** because it operates on two operands.
- All other arithmetic operators except the **minus** are also binary.
- A **unary operator** operates on one operand as in,

  –x

- In the expression below, the right operand for the addition operator is itself an expression. Often a nested expression is called a **subexpression**.

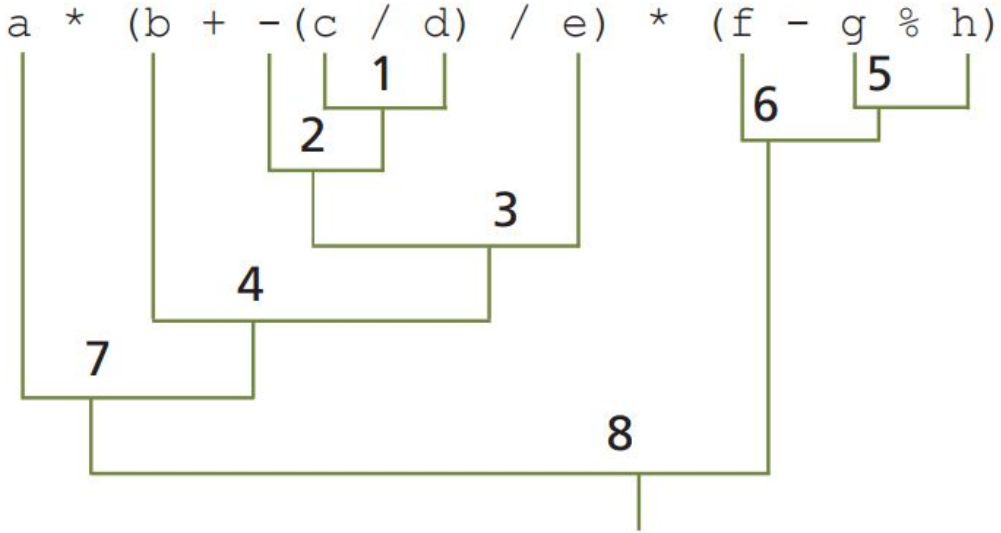
  x + 3 * y

Arithmetic Expressions

- When two or more operators are present in an expression, we determine the order of evaluation by following the **precedence rules**.
  - **Subexpression** **()** **First**
  - **Unary operator** **-, +** **Second**
  - **Multiplicative operator** ***, /, %** **Third**
  - **Additive operator** **+, -** **Fourth**

Arithmetic Expressions

- The following example illustrates the precedence rules applied to a complex arithmetic expression:



Arithmetic Expressions

- When the data types of variables and constants in an arithmetic expression are different data types, then a **casting conversion** will take place.
- A **casting conversion**, or **typecasting**, is a process that converts a value of one data type to another data type.
- Two types of casting conversions in Java are **implicit** and **explicit**.
- An **implicit conversion** called **numeric promotion** is applied to the operands of an arithmetic operator.
- This conversion is called **promotion** because the operand is converted from a lower to a higher precision.
  - **10 / 4.0 = 2.5**

Arithmetic Expressions

- Instead of relying on implicit conversion, we can use explicit conversion to convert an operand from one data type to another.
- Explicit conversion is applied to an operand by using a typecast operator.
- For example, to convert the int variable x in the expression, x / 3 to float we apply the typecast operator **(float)** as,
  - **(float) x / 3**

Arithmetic Expressions

- An **assignment conversion** is a type of implicit conversion that occurs when you assign a value to a variable of a different data type, the value is converted to the data type of the variable.
- An assignment conversion occurs only if the data type of the variable has a higher precision than the data type of the expression's value. For example,

```
double number;
number = 25;
```

is valid, but

```
int number;
number = 234.56;
```

is not.

Arithmetic Expressions

- A **wrapper** class is a class that encapsulates, or "wraps," a primitive data type into an object.
- The Java programming language primarily uses primitive data types like int, char, float, etc., to represent simple values.
- However, in certain situations, you might need to work with objects rather than primitives.
  - Eg: Java collections (e.g., ArrayList, LinkedList, HashSet) can only store objects, not primitive types.
- This is where wrapper classes come in. Wrapper classes allow you to treat primitive data types as objects.

Arithmetic Expressions

```
public class WrapperExample {
    public static void main(String[] args) {
        // Primitive int
        int primitiveInt = 42;

        // Using Integer wrapper class
        Integer wrappedInt = Integer.valueOf(primitiveInt); // Wrapping the primitive int

        // Performing operations with the wrapper
        int result = wrappedInt + 10; // Unboxing - Integer to primitive int
        System.out.println("Result: " + result);

        // Converting a String to an Integer
        String numberStr = "12345";
        Integer parsedInt = Integer.parseInt(numberStr);
        System.out.println("Parsed Integer: " + parsedInt);

        // Converting back to a String
        String intToStr = wrappedInt.toString();
        System.out.println("Integer to String: " + intToStr);
    }
}
```

Arithmetic Expressions

- we often have to increment or decrement the value of a variable by a certain amount. For example, to increase the value of sum by 5, we write

```
sum = sum + 5;
```

- We can rewrite this statement without repeating the same variable on the left- and right-hand sides of the assignment symbol by using the **shorthand assignment** operator:

```
sum += 5;
```

Constants

- If we want a value to remain fixed, then we use a constant. A constant is declared in a manner similar to a variable but with the additional reserved word final.

```
final double PI = 3.14159;

final short FARADAY_CONSTANT = 23060;
final double CM_PER_INCH = 2.54;

final int MONTHS_IN_YEAR = 12;
```

Sample Java Standard Classes

- **Standard Output**
  - When a program computes a result, we need a way to display this result to the user of the program.
  - One of the most common ways to do this in Java is to use the console window.
  - The console window is also called the standard output window.
  - We output data such as the computation results or messages to the console window via **System.out**.

Sample Java Standard Classes

- **Standard Output**
  - The System class in Java provides several helpful class data values.
  - Among them is a static field called **out,** which is an instance of the PrintStream class.
  - Any data that we send to the **System.out** object will be displayed on the console window.
  - To output a value using the standard output, we can utilize the **print** or **println** method.

Quick questions

1. Write a Java statement to display the text **I Love Java** in the console window.

2. Write statements to display the following shopping list in the console window. Don't forget to include blank spaces so the item names appear indented.

Shopping List:

Apple

Banana

Low-fat Milk

Sample Java Standard Classes

- **String**
  - The textual values we passed to the print method are instances of the **String** class.
  - A sequence of characters separated by double quotes is String constants.
  - As **String** is a class, we can create an instance and give it a name. For example,

```
String country;
country = new String("Sri Lanka");
```

Sample Java Standard Classes

- **String**
  - Unlike in other classes, the explicit use of new to create an instance is optional for the String class.
  - We can create a new **String** object, for example, in this way:

```
String country;
country= "Sri Lanka";
```

  - There are close to 50 methods defined in the **String** class. For example,
    - **substring**
    - **length**
    - **indexOf**

Sample Java Standard Classes

- **String**
  - We can extract a **substring** from a given string by specifying the beginning and ending positions. For example,

```
String country;
country = "Sri Lanka";
System.out.print(country.substring(0, 3));
```

  - We can find out the number of characters in a **String** object by using the length method. For example,

```
System.out.print(country.length());
```

Sample Java Standard Classes

- **String**
  - We can extract a **substring** from a given string by specifying the beginning and ending positions. For example,

```
String country;
country = "Sri Lanka";
System.out.print(country.substring(0, 3));
```

  - To locate the index position of a substring within another string, we use the indexOf method. For example,

```
System.out.print(country.indexOf("Lanka");
```

Quick questions

1. What will be the output of this program.

```
String java;
java = "I Love Java and Java loves me.";
System.out.println(java.indexOf(love));
```

2. What will be the output of this program.

```
String country;
country = "Sri Lanka";
String java;
java = "I Love Java and Java loves me.";
System.out.println(country+"! "+java);
```

Sample Java Standard Classes

- **Standard Input**
  - As we have **System.out** for output, we have **System.in** for input.
  - We call the technique to input data using **System.in** standard input.
  - **System.in** accepts input from the keyboard.
  - **System.in** is an instance of the **InputStream** class that provides only a facility to input 1 byte at a time with its **read** method.

Sample Java Standard Classes

- **Standard Input**
  - The **Scanner** class from the **java.util** package provides a necessary input facility to accommodate various input routines.
  - To input data from the standard input by using a **Scanner** object, we first create it by passing **System.in** as follows:

```
import java.util.*;

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

Standard Input example

```
import java.util.*;
class SampleScanner {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String firstName;
        System.out.print("Enter your first name: ");
        firstName = scanner.next( );
        System.out.println("Nice to meet you, " + firstName +".");
    }
}
```

Quick question

1. Write a program the get first name and last name of a user and print their full name as output.

2. Write an application that asks for the user's first, middle, and last names and replies with the user's initials.

Displaying Numerical Values

- We can use the print and println methods to output numerical values:

```
int num = 15;
System.out.print(num);
```

- By using the concatenation operation, it is possible to output multiple values with a single print or println method:

```
System.out.print(30 + " " + 40);
```

- The same plus symbol we used for concatenation can be used to add numerical values, for example,

```
System.out.print(30 + 40);
```

Displaying Numerical Values

- The plus symbol, therefore, could mean two different things:
  - **String concatenation**
  - **Numerical addition**
- When a symbol is used to represent more than one operation, this is called **operator overloading**.
- When the Java compiler encounters an overloaded operator, the compiler determines the meaning of a symbol by its context.
- If the left operand and the right operand of the plus symbol are both,
  - **Numerical values**, then the compiler will treat the symbol as addition;
  - **Otherwise**, it will treat the symbol as concatenation.

Displaying Numerical Values

- The plus symbol operator is evaluated from left to right therefore,

```
int x = 1;
int y = 2;
System.out.print("test" + x + y);
```

- will result in output being set to **test12**. while the statement,

```
System.out.print(x + y + "test");
```

- will result in output being set to **3test**.
- What will be the output in the following statement,

```
System.out.print("test" + (x + y));
```

Getting Numerical Input

- To input strings, we've used the **next** or **nextLine** method of the Scanner class.
- For the numerical input values, we use an equivalent method that corresponds to the data type of the value we try to input.
- For instance, to input an int value, we use the **nextInt** method.

```
Scanner scanner = new Scanner(System.in);
int age;
System.out.print("Enter your age: ");
age = scanner.nextInt( );
System.out.println("Your are " + age + "
years old");
```

Getting Numerical Input #1

```
class Person{
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int height;
        float gpa;
        System.out.print("Enter your height in inches: ");
        height = scanner.nextInt( );
        System.out.print("Enter your gpa: ");
        gpa = scanner.nextFloat( );
    }
}
```

Thank you