**CHAPTER 2: ELEMENTARY PROGRAMMING**

* 1. **READING INPUT FROM THE CONSOLE**

Reading input from the console enables the program to accept input from the user.

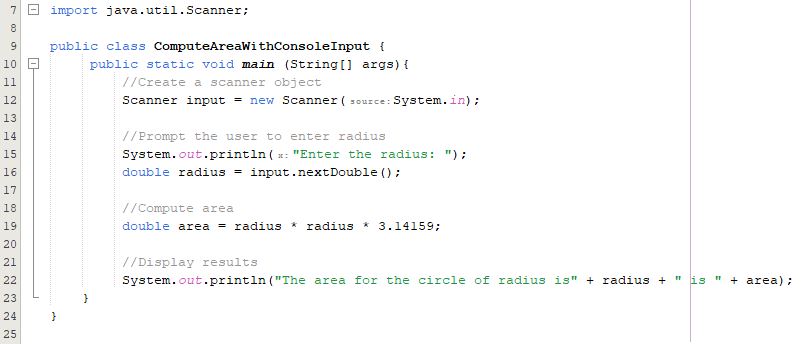
You can use the **Scanner** class for console input. Java uses **System.out** to refer to the standard output device, and **System.in** to the standard input device.

By default, the output device is the display monitor, and the input device is the keyboard.

To perform console input, you need to use the Scanner class to create an object to read input from System.in, as follows:

**Scanner input = new Scanner(System.in);**

An object may invoke its methods. To invoke a method on an object is to ask the object to perform a task. You can invoke the nextDouble() method to read a double value as follows: **double variablename = input.nextDouble();**

**Example:** 

In this figure it is shown how to read input from console.

* 1. **IDENTIFIERS**

Identifiers are the names that identify the elements such as classes, methods, and variables in a program.

All identifiers must obey the following rules:

* An identifier is a sequence of characters that consists of letters, digits, underscores (\_), and dollar signs ($).
* An identifier must start with a letter, an underscore (\_), or a dollar sign ($). It cannot start with a digit.
* An identifier cannot be a reserved word. See Appendix A for a list of reserved words. Reserved words have specific meaning in the Java language. Keywords are reserved words.
* An identifier can be of any length.

**NOTE:** Since Java is case sensitive, **area, Area**, and **AREA** are all different identifiers.

* 1. **VARIABLES**

Variables are used to represent values that may be changed in the program.

Variables are for representing data of a certain type. To use a variable, you declare it by telling the compiler its name as well as what type of data it can store.

The syntax for declaring a variable is:

**datatype variableName;**

These examples use the data types: **int and double, byte, short, long, float, char, and Boolean.**

If variables are of the same type, they can be declared together, as follows:

**datatype variable1, variable2, ..., variablen;**

* 1. **ASSIGNMENTS STATEMENTS AND ASSIGNMENTS EXPRESSIONS**

An assignment statement assigns a value to a variable. An assignment statement can also be used as an expression in Java.

After a variable is declared, you can assign a value to it by using an assignment statement. In Java, the equal sign (=) is used as the assignment operator. The syntax for assignment statements is as follows:

**variable = expression;**

* 1. **NAMED CONSTANTS**

A named constant is an identifier that represents a permanent value.

The value of a variable may change during the execution of a program, but a named constant, or simply constant, represents permanent data that never changes.

Here is the syntax for declaring a constant: **final datatype CONSTANTNAME = value;**

* 1. **NAMED CONVENTIONS**

Sticking with the Java naming conventions makes your programs easy to read and avoids errors.

Make sure you choose descriptive names with straightforward meanings for the variables, constants, classes, and methods in your program.

Listed below are the conventions for naming variables, methods, and classes:

* Use lowercase for variables and methods
* Capitalize the first letter of each word in a class name
* Capitalize every letter in a constant, and use underscores between words

**NOTE:** Do not choose class names that are already used in the Java library. For example, since the **System** class is defined in Java, you should not name your class **System**.

* 1. **NUMERIC DATA TYPES AND OPERATIONS**

Java has six numeric types for integers and floating-point numbers with operators

**+ , - , \* , / , and %.**

Every data type has a range of values. The compiler allocates memory space for each variable or constant according to its data type.

**Reading numbers from the keyboard**

* nextByte() reads an integer of the byte type.
* nextShort() reads an integer of the short type.
* nextInt() reads an integer of the int type.
* nextLong() reads an integer of the long type.
* nextFloat() reads a number of the float type.
* nextDouble() reads a number of the double type.

Example:

**System.out.print("Enter an int value: ");**

**int intValue = input.nextInt();**

**Numeric operators**

The operators for numeric data types include the standard arithmetic operators: **addition (+), subtraction (–), multiplication (\*), division (/), and remainder (%).**

**EXPONENT OPERATIONS**

The **Math.pow(a, b)** method can be used to compute **a^b**. The **pow** method is defined in the **Math** class in the Java API.

Example: **System.out.println(Math.pow(2, 3));** **This will display 8.**

* 1. **NUMERIC LITERALS**

A literal is a constant value that appears directly in a program.

For example, 34 and 0.305 are literals in the following statements:

**int numberOfYears = 34;**

**double weight = 0.305;**

**INTEGER LITERALS**

An integer literal can be assigned to an integer variable as long as it can fit into the variable.

**NOTE:** The statement **byte b = 128**, for example, will cause a compile error, because **128** cannot be stored in a variable of the **byte** type. (Note the range for a byte value is from –128 to 127.).

**FLOATING-POINT LITERALS**

Floating-point literals are written with a decimal point. By default, a floating-point literal is treated as a double type value.

For example, **5.0** is considered a **double** value, not a **float** value.

* 1. **JSHELL**

JShell is a command line tool for quickly evaluating an expression and executing a statement.

Shell is a command line interactive tool introduced in Java 9. JShell enables you to type a single Java statement and get it executed to see the result right away without having to write a complete class.

* Open a Command Window and type jshell to launch JShell.
* You can enter a Java statement from the jshell prompt.
* To print the variable, simply type **x**, Alternatively, you can type **System.out.println(x).**
* You can list all the declared variables using the **/vars**.
* In JShell, if you don’t specify a variable for a value, JShell will automatically create a variable for the value.
* To exit JShell, enter /exit.
  1. **EVALUATING EXPRESSIONS AND OPERATOR PRECEDENCE**

Java expressions are evaluated in the same way as arithmetic expressions.

Writing a numeric expression in Java involves a straightforward translation of an arithmetic expression using Java operators.

Example, the arithmetic expression:

Can be translated into a Java expression as follows:

**(3 + 4 \* x) / 5 – 10 \* (y - 5) \* (a + b + c) / x + 9 \* (4 / x + (9 + x) / y);**

* 1. **CASE STUDY: DISPLAYING THE CURRENT TIME**

You can invoke **System.currentTimeMillis()** to return the current time.

The problem is to develop a program that displays the current time in GMT (Greenwich Mean Time) in the format **hour:minute:second,** such as **13:19:8.**

* 1. **AUGMENTED ASSIGNMENT OPERATORS**

The operators +, -, \*, /, and % can be combined with the assignment operator to form augmented operators.

Java allows you to combine assignment and addition operators using an augmented (or compound) assignment operator. For example, the preceding statement can be written as

**count += 1;**

The **+=** is called the addition assignment operator.

Augmented Assignment

* += Addition assignment.
* -= Subtraction assignment.
* \*= Multiplication assignment.
* /= Division assignment.
* %= Remainder assignment.

**NOTE:** There are no spaces in the augmented assignment operators.

Example, + = should be **+=**.

* 1. **INCREMENT AND DECREMENT OPERATORS**

The increment operator **(++)** and decrement operator **(--)** are for incrementing and decrementing a variable by 1.

For example, the following code increments i by 1 and decrements j by 1.

int i = 3, j = 3;

i++; 🡪 i becomes 4

j——; 🡪 j becomes 2

Increment and Decrement Operator

* ++var
* var++
* --var
* var—
  1. **SOFTWARE DEVELOPMENT PROCESS**

The software development life cycle is a multistage process that includes **requirements specification, analysis, design, implementation, testing, deployment, and maintenance.**

**MAINTENANCE**

**TESTING**

**DEPLOYMENT**

**IMPLEMENTATION**

**SYSTEM DESIGN**

**SYSYEM ANALYSIS**

**REQUIREMENTS**

**NOTE: At any stage of the software development life cycle!.**

* 1. **COMMON ERRORS AND PITFALLS**

Common elementary programming errors often involve undeclared variables, uninitialized variables, integer overflow, unintended integer division, and round-off errors.

* **Common Error 1: Undeclared/Uninitialized Variables and Unused Variables**

**double interestRate = 0.05;**

**double interest = interestrate \* 45;**

* **Common Error 2:** Integer Overflow

Numbers are stored with a limited numbers of digits.

**int value = 2147483647 + 1;**

* **Common Error 3: Round-off Errors**

A round-off error, also called a rounding error, is the difference between the calculated approximation of a number and its exact mathematical value.

**System.out.println(1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);** displays 0.5000000000000001 not 0.5

**System.out.println(1.0 - 0.9);** displays 0.09999999999999998, not 0.1

* **Common Error 4: Unintended Integer Division**

Java uses the same divide operator **/** to perform both integer and floating-point division. When two operands are integers, the **/** operator performs an integer division

**int number1 = 1;**

**int number2 = 2;**

**double average = (number1 + number2) / 2;**

**System.out.println(average); This will display 1**

**While**

**int number1 = 1;**

**int number2 = 2;**

**double average = (number1 + number2) / 2.0;**

**System.out.println(average); This will display 1.5**

**THE END!!**