|  |  |
| --- | --- |
| **Topic** | **Input, Output, Java Operators and Expressions** |
| **Learning Outcomes** | We Recognized the importance of seeing the result of our hard works and be able to process data that comes from a user input, Java provides classes that allows us to get input from the keyboard process them and be able to display the result.  At the end of the lesson, you will be able to:   1. Use the classes intended for Input and output 2. Create Java Expressions 3. Performs Computations 4. Understand the use of Java Operators. 5. Know the rule of operator precedence 6. Identify the different Math class methods |
| **References** | Lemay, L., Perkins, C. L. (1996), Teach Yourself JAVA in 21 days,  Indianapolis, Indiana, Sams.net.  Doug Lowe, Java® All-in-One For Dummies®, 4th Edition), New Jersey,  John Wiley & Sons, Inc.  Barry Burd Ph.D., Java® For Dummies®, 7th Edition, New Jersey,  John Wiley & Sons, Inc.  David J. Eck, Introduction to Programming Using Java Version 5.0, December 2006  Geneva, NY, Hobart and William Smith Colleges  Jester Lhee I. Pandio, Computer Programming 1, Global Port Taguig City, STI College |

**Discussion**

**Input and Output**

**Input and output**, or I/O is the communication between an information processing system, such as a computer, and the outside world, possibly a human or another information processing system. Inputs are the signals or data received by the system and outputs are the signals or data sent from it

**The Scanner Class**

Until Java 1.5, getting text input from the user in a console-based Java program was not easy, but with Java 1.5, a new class called Scanner was introduced, to simplify the task of getting input from the user.

A class in the java.util package. The methods of this class are used to read data from the standard input device and store data into variables.

**Importing the Scanner** **class**

Before you can use the Scanner class in a program, you must import it. To do that, you code an import statement at the beginning of the program, before the class declaration

***import java.util.Scanner;***

Note that java and util are not capitalized, but Scanner is.

**Declaring and creating a Scanner object**

Before you can use the Scanner class to read input from the console, you must declare a Scanner variable and create an instance of the Scanner class as shown below.

static Scanner read = new Scanner(System.in);

To create a Scanner object, you use the new keyword followed by a call to the Scanner class constructor. Note that the Scanner class requires a parameter that indicates the input stream where the input comes from. The ***System.in*** indicates that the input will come from the keyboard.

**Getting input**

To read an input value from the user, you can use one of the methods of the Scanner class that are listed in Table below.

Table 4.1: Listings of methods of the scanner class

|  |  |  |
| --- | --- | --- |
| Method | Description | Example |
| nextDouble() | Retrieves input as a double | Double x = objectName.nextDouble(); |
| nextInt() | Retrieves input as an int | int x = objectName.nextInt(); |
| nextLine() | Retrieves the next line of data and returns it as a String | String x = objectName.nextLine(); |
| next() | Retrieves the next complete token as a String | String x = objectName.next(); |
| nextShort() | Retrieves input as a short | short x = objectName.nextShort(); |
| nextByte() | Retrieves input as a byte | byte x = objectName.nextByte(); |
| nextFloat() | Retrieves input as a float | float x = objectName.nextFloat(); |
| nextLong() | Retrieves input as n long | long x = objectName.nextLong(); |

The Scanner class does not contain a next method for the char data type. To retrieve a single character from the keyboard, use the next () method, then use the charAt([index]) method. For example, input.next().charAt(0), the next() method will return the token of the input as a String and the charAt(0) method returns the first character in the String through index 0.

**Getting Input with the JOptionPane Class**

If you are using a version of Java before Java 1.5, you donot have the luxury of using the Scanner class to read input directly from the user via a console window. You can use the JOptionPane class, however, to display simple dialog boxes such as the one shown below to get text input from the user. Then you can use the parse methods of the primitive-type wrapper classes to convert the text entered by the user to the appropriate primitive type.



The JOptionPane class is a part of the javax.swing package, so you need to add an import javax.swing.JOptionPane statement to the beginning of any program that uses this class.

**Program Listings 4.1**

import javax.swing.JOptionPane;

public class DialogApp{

public static void main(String[] args) {

String s;

s = JOptionPane.showInputDialog("Enter an integer:");

int x = Integer.parseInt(s);

System.out.println("You entered " + x + ".");}}

**Printing Data with System.out**

Standard output is a stream designed to display text output onscreen. When you run a Java program under Windows, a special console window opens, and the standard output stream is connected to it. Then any text you send to standard output is displayed in that window

Examples of these output statements are the following:

System.out.print (“This is a text.”);

and

System.out.println (“This is a text.”);

* The print () method displays an output, and the insertion point stays in the current line.
* The println () method moves the insertion point to the following line after the output is displayed.

When displaying a string with variables, simply concatenate them using the plus (+) sign.

For example:

String name = “Juan Dela Cruz”;

System.out.println (“Hello! Welcome” + name + “.”);

and

int kilometer = 4;

System.out.println (“Distance Travelled is: “+ kilometer + “kilometer.”);

The output of print and println methods are in the format of strings.

Java includes a method named printf () that can be used to create an output in a specific format. This method is used to add formatting instructions that specify the output before displaying it on the display screen, such as modifying the number of digits to include after a decimal point.

Example, consider the following:

double price = 19.5;

System.out.println (“Price using println :” + price);

System.out.printf (“Price using print formatting: %.2f”, price);

The above produce the following output:

Price using println: 19.5

Price using printf formatting: 19.50

Using the printf method, the double type variable is formatted to have two (2) decimal places. In this example, the first argument to %.2f is a string called the format specifier, and the second argument is the value to be output in that format. A format specifier is a placeholder for a numeric value.

The format specifier %.2f tells the JVM to return the variable with two (2) digits after the decimal point. The %.2f means the output is a floating-point number.

**Table 4.2: Listings of format used for the printf method**

|  |  |  |
| --- | --- | --- |
| Format Specifier | Type of output | Example |
| %c | Character | A single character: %c |
| A single character in a field of two (2) spaces: %2c |
| %d | Decimal integer number | An integer |
| %f | Floating-point number | A floating-point number: %f |
| With 2 digits after the decimal point: %.2f |
| With 2 digits after the decimal in a field of 6 spaces: %6.2f |
| %e | Exponential floating-point number | A floating-point number in exponential format: %e |
| %s | String | A string formatted to a field of 10 spaces: %10s |
| With first 2 characters of the string: %.2s |

**Java Operators**

Java Operators or simply operators is a special symbol or keyword that’s used to designate a mathematical operation or some other type of operation that can be performed on one or more values, called operands.

1. **Arithmetic Operators**

|  |  |  |
| --- | --- | --- |
| Table 4.3. Java Arithmetic Operators. | | |
| Operator | **Description** | **Description** |
| + | **Addition** | **Add or sum up two numbers** |
| - | **Subtraction** | **Subtract two numbers** |
| \* | **Multiplication** | **Multiply two numbers** |
| / | **Division** | **Divide two numbers** |
| % | **Modulus Division** | **Get the remainder after dividing the numbers** |
| ++ | **Increment** | **Add 1 to the value** |
| --- | **Decrement** | **Subtract 1 to the value** |

**Table 4.4: Arithmetic Operation Examples**

|  |  |
| --- | --- |
| **Integer Operations** | **Double operations** |
| int a = 32, b = 5; | double x = 5.5, y = 2.0; |
| int c = a + b; // c is 37 | double m = x + y; // m is 7.5 |
| int d = a - b; // d is 27 | double n = x - y; // n is 3.5 |
| int f = a / b; // f is 5 (32 / 5 is 6 remainder 2) | double o = x \* y; //o is 11.0 |
| int g = a % b; // g is 2 (32 / 5 is 6 remainder 2) | double q = x % y; // q is 1.5 |
| a++; // a is now 33 | x++; // x is now 6.5 |
| b--; // b is now 4 | y--; // y is now 1.0 |

When you divide two int values, the result is an integer value, even if you assign it to a double variable.

For example:

int a = 21, b = 6;double answer = a / b; // answer = 3

If that is not what you want, you can cast one of the operands to a double before performing the division. (Casting an operand means converting its value from one data type to another).

Here is how:

int a = 21, b = 6;double answer = (double)a / b; // answer = 3.5

**Program Listing 4.2**

**Sample Program**

**import java.util.\*;**

**public class AreaOfCircle {**

**public AreaOfCircle() { // Constructor**

Scanner input = new Scanner(System.in);

final double PI = 3.1416;

double radius;

double area;

String geometricShape;

System.out.print("Enter the radius of the circle : ");

radius = input.nextDouble();

System.out.print("Geometric Type : "); geometricShape = input.next();

***area = PI \* radius\*radius;***

System.out.println("The area of the circle is : " + area);

System.out.printf("The area of the circle is : %.2f \n", area);

System.out.printf("With the radius of : %.2f \n", radius);

System.out.printf("Geometric Type is : %10s \n", geometricShape);

System.out.printf("The area of the circle in exponential form is : %.2e \n", area);

System.out.printf("The area of the circle in exponential form is : %e", area);

}

**public static void main(String[] args){**

**new AreaOfCircle();**

**}**

**}**

**Challenge:**

Compile the program above and analyzed its output, what have you noticed?

1. **Relational Operators(Comparison)**

Java has several expressions for testing equality and magnitude. All of these expressions return a boolean value (that is, true or false). Table 5 shows the relational operators:

**Table 5.5. Relational Operators**

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Example |
| == | Equal | x == 3 |
| != | Not Equal | x!=3 |
| < | Less than | x < 3 |
| > | Greater than | x > 3 |
| <= | Less than or equal to | x <= 3 |
| >= | Greater than or equal to | x >= 3 |

1. **Logical and Bitwise Operators**

**Table 5.6. Logical Operators**

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Example |
| && or & | Logical AND | (x>3 && y <3) |
| || or | | Logical OR | (x>3 || y <3) |
| ! | Logical NOT | !(x < 3) |
| ^ | Logical XOR | (x>3 ^ y <3) |
|  |  |  |
|  |  |  |

**Note:**

1. The Logical operators often expressed as conditional expressions and are being applied to branching and iterations
2. Logical & (And) Logical |(OR) must have boolean as operands to process as logical operators otherwise it will used as a bitwise operator.

**Table 4.7. Bitwise Operators**

|  |  |  |  |
| --- | --- | --- | --- |
| Operator | Meaning | Example  int a=12, b=9; | Result |
| & | Bitwise AND | int c = a&b; | c = 8; |
| | | Bitwise OR | int c = a|b; | c = 13; |
| ^ | Bitwise XOR | int c = a^b; | c = 5; |
| << | Left shift | int c = a<<b; | c = 6144; |
| >> | Right shift | int c = a>>b; | c = 0; |
| <<< | Zero fill right shift | int c = a>>>b; | c = 0; |
| ~ | Bitwise complement | int c = a~; | c = -13; |

**How to:**

**Bitwise AND (&)**

int a = 12; int b = 9;

int c = a & b; // result is 8;

a = 12 🡪 1100 (Binary 12)

b = 9 🡪 1001 (Binary 9)

c = 8 1000 (Binary AND result)

**Bitwise OR ( | )**

int a = 12; int b = 9;

int c = a | b; // result is 13;

a = 12 🡪 1100 (Binary 12)

b = 9 🡪 1001 (Binary 9)

c = 13 1101 (Binary OR result)

**Bitwise XOR (^)**

int a = 12; int b = 9;

int c = a & b; // result is 5;

a = 12 🡪 1100 (Binary 12)

b = 9 🡪 1001 (Binary 9)

c = 5 0101 (Binary XOR result)

**Bitwise Left Shift (<<)**

int a = 12; int b = 9;

int c = a & b; // result is 6144;

a = 12 🡪 1100 (Binary 12) number to be shifted

b = 9 🡪 1001 (Binary 9) number of times the operand will be shifted

c = 6144 1100 0000 0000 0 (Binary Left Shift result) // shifted 9 times to the left

**Bitwise Right Shift (<<)**

int a = 12; int b = 9;

int c = a & b; // result is 0;

a = 12 🡪 1100 (Binary 12) number to be shifted

b = 9 🡪 1001 (Binary 9) number of times the operand will be shifted

c = 0 00 0000 0000 0 (Binary Left Shift result) // shifted 9 times to the right

Note: in shifting the binary numbers to the right by more than or equal to the number of bits of itself will result to a zero value.

Ex.

If the number of bits to be shifted is four, and it will be shifted four or more

times, the result will be zero.

**Zero fill right shift (<<<)**

int a = 12; int b = 9;

int c = a & b; // result is 0;

a = 12 🡪 1100 (Binary 12) number to be shifted

b = 9 🡪 1001 (Binary 9) number of times the operand will be shifted

c = 0 00 0000 0000 0 (Binary Left Shift result) // shifted 9 times to the right

**Bitwise complement** (~)

int a = 12; int b = 9;

int c = a ~; // result is -13;

a = 12 🡪 1100 (Binary 12)

0000 1100 (this is an 8 bit binary 12)

a = -13🡪 1111 0011 (flip the bits, one’s complement) // this is now the -13

to check the result do a two’s complement to derive the positive value of 13

**Program Listings 4.3**

**Sample Program**

public class BitwiseOperators {

public static void main(String[] args){

int a=12, b=9, c=2;

System.out.println(a&b);

System.out.println(a|b);

System.out.println(a^b);

System.out.println(a<<b);

System.out.println(a>>b);

System.out.println(a>>>b);

System.out.println(~b);

if(a == 5 & b==10){

System.out.println(a|b);

}else if (a == 5 & b==10){

System.out.println(a|b);

**}**

**}**

**}**

**4. Assignment Operators**

**Table 4.8 Assignment Operator**

|  |  |  |  |
| --- | --- | --- | --- |
| Operator | Meaning | Example  int a=12, b=9; | Result |
| += | Addition and assignment | int a+=10; 🡪 a = a + 10; | c = 22; |
| -= | Subtraction and assignment | int a+=10; 🡪 a = a – 10; | c = 2; |
| \*= | Multiplication and assignment | int a\*=10; 🡪 a = a \* 10; | c = 120; |
| /= | Division and assignment | int a/=10; **🡺**a = a / 10; | c = 1; |
| %= | Remainder and assignment | int a%10; 🡺 a = a % 10; | c = 2; |

**Program Listings 4.4**

public class AssignmentOperators {

public static void main(String[] args){

int a=12, b=9, c=2;

System.out.println("Addition and Assignment : " + (a+=10));

a=12;

System.out.println("Subtraction and Assignment : " + (a-=10));

a=12;

System.out.println("Multiplication and Assignment : " +(a\*=10));

a=12;

System.out.println("Division and Assignment : " + (a/=10));

a=12;

System.out.println("Remainder and Assignment : " +(a%=10));

}

}

**Categorizing operators by the number of operands**

A common way to categorize Java’s operators is to use the number of operands the operator works on. When the operators are categorized this way, you find three types:

1. **Unary operators**

Operators that work on just one operand. Examples of unary operators are negation (-x, which returns the negative of x) and increment (x++, which adds 1 to x).A unary operator can be a prefix operator or a postfix operator. A prefix operator is written before the operand, like this:

operator operand

A postfix operator is written after the operand, like this:

operand operator

1. **Binary operators**

Operators that work on two operands. Examples of binary opera-tors are addition (x + y), multiplication (invoiceTotal \* taxRate), and comparison operators (x < leftEdge). In Java, all binary operators are infix operators, which means they appear between the operands, like this:

operand1 operator operand2

1. **Ternary operators**

Operators that work on three operands. Java has only one ternary operator, called the conditional operator(?:). The conditional operator is also infix:

operand1 ? operand2 : operand3

**The Math Class**

Java’s built-in operators are useful, but they do not come anywhere near providing all the mathematical needs of most Java programmers. That is where the Math class comes in. It includes a bevy of built-in methods that perform a wide variety of mathematical calculations, from basic functions such as calculating an absolute value or a square root to trigonometry functions such as sin and cos (sine and cosine), to practical functions such as rounding numbers or generating random numbers.

All the methods of the Math class are declared as static methods, which means you can use them by specifying the class name Math followed by a period and a method name.

Here is a statement that calculates the square root of a number stored in a variable named y:

double x = Math.sqrt(y);

**Constants of the Math class**

|  |  |  |
| --- | --- | --- |
| **Table 4.9 Math Class constant** | | |
| **Constant** | **What it is** | **Value** |
| PI | The constant pi (π), the ratio of a circle’s radius and diameter | 3.141592653589793 |
| E | The base of natural logarithms | 2.718281828459045 |

Note that these constants are only approximate values, because both π and e are irrational numbers

**Program Listings 4.5**

import java.util.Scanner;

public class CircleAreaApp{

static Scanner sc = new Scanner(System.in);

public static void main(String[] args) {

System.out.println("Welcome to the circle area calculator.");

System.out.print("Enter the radius of your circle: ");

double r = sc.nextDouble();

double area = Math.PI \* (r \* r);

System.out.println("The area is " + area);

}

}

|  |  |
| --- | --- |
| Table 4.9 Mathematical functions provided by the Math Class | |
| Method | Explanation |
| abs(argument) | Returns the absolute value of the argument. The argument can be an int, long, float, or double. The return value is the same type as the argument |
| cbrt(argument) | Returns the cube root of the argument. The argument and return value are doubles. |
| exp(argument) | Returns the natural logarithm (base e) of the argument. The argument and the return value are doubles. |
| log10(argument) | Returns the base 10 logarithm of the argument. The argument and the return value are doubles. |
| max(arg1, arg2) | Returns the larger of the two arguments. The arguments can be int, long, float, or double, but both must be of the same type. The return type is the same type as the arguments. |
| min(arg1, arg2) | Returns the smaller of the two arguments. The arguments can be int, long, float, or double, but both must be of the same type. The return type is the same type as the arguments. |
|  |  |
| Method | Explanation |
| pow(arg1, arg2) | Returns the value of the first argument raised to the power of the second argument. Both arguments and the return value are doubles. |
| random() | Returns a random number that is greater than or equal to 0.0 but less than 1.0. This method doesn’t accept an argument, but the return value is a double |
| signum(argument) | Returns a number that represents the sign of the argument: –1.0 if the argument is negative, 0.0 if the argument is zero, and 1.0 if the argument is positive. The argument can be a double or a float. The return value is the same type as the argument. |
| sqrt(argument) | returns the square root of the argument. The argument and return value are doubles. |

**Program Listings 4.6**

public class MathClass {

public static void main(String[] args){

int b = 10, a =20;

System.out.println(Math.abs(b));

System.out.println(Math.cbrt(b));

System.out.println(Math.exp(b));

System.out.println(Math.hypot(a,b));

System.out.println(Math.log(a));

System.out.println(Math.log10(a));

System.out.println(Math.max(a, b));

System.out.println(Math.min(a, b));

System.out.println(Math.pow(a, b));

System.out.println(Math.random());

System.out.println(Math.signum(b));

System.out.println(Math.sqrt(a));

}

}

**Creating Random Numbers**

**Program Listings 4.7**

public class RandomNumbers {

public RandomNumbers() {

int low = 1; // the lowest value in the range

int high = 6; // the highest value in the range

int rnd = (int)(Math.random() \* (high - low + 1)) + low;

System.out.println("Random Number Generated : " + rnd);

}

public static void main(String[] args){

new RandomNumbers();

}

}

**Rounding functions**

|  |  |
| --- | --- |
| Table 4.10 Rounding Functions | |
| Method | Explanation |
| ceil(argument) | Returns the smallest double value that is an integer and is greater than or equal to the value of the argument. |
| floor(argument) | Returns the largest double value that is an integer and is less than or equal to the value of the argument. |
| rint(argument) | Returns the double value that is an integer and is closest to the value of the argument. If two integer values are equally close, it returns the one that is even. If the argument is already an integer, it returns the argument value |
| round(argument) | Returns the integer that is closest to the argument. If the argument is a double, it returns a long. If the argument is a float, it returns an int. |

**Program Listing 4.9**

import static java.lang.System.out;

public class RoundingNumbers {

public static void main(String[] args) {

double x = 29.4, y = 93.9, z = -19.3;

out.println(Math.round(x));

out.println(Math.round(y));

out.println(Math.round(z));

out.println(Math.ceil(x));

out.println(Math.ceil(y));

out.println(Math.ceil(z));

out.println(Math.floor(x));

out.println(Math.floor(y));

out.println(Math.floor(z));

out.println(Math.rint(x));

out.println(Math.rint(y));

out.println(Math.rint(z));

}

}

**Operator Precedence**

Operator precedence determines the order in which expressions are evaluated. This, in some cases, can determine the overall value of the expression

For example, take the following expression:

y = 6 + 4 / 2

Depending on whether the 6 + 4 expression or the 4 ÷ 2 expression is evaluated first, the value of y can end up being 5 or 8. Operator precedence determines the order in which expressions are evaluated, so you can predict the outcome of an expression

|  |  |
| --- | --- |
| Table 4.11 Operator Precedence | |
| Operator | Notes |
| . [] () | Parentheses () group expressions; dot (.) is used for access to methods and variables within objects and classes; [] is used for arrays |
| ++ –– ! ~ instanceof | Returns true or false based on whether the object is an instanceof the named class or any of that class’s super classes |
| new (type)expression | The new operator is used for creating new instances of classes; () in this case is for casting a value to another type. |
| \* ÷ % | Multiplication, division, modulus |
| + – | Addition, subtraction |
| << >> >>> | Bitwise left and right shift |
| < > ≤ ≥ | Relational comparison tests |
| == != | Equality |
| & | AND |
| ^ | XOR |
| | | OR |
| && | Logical AND |
| || | Logical OR |
| ?: | Shorthand for if... then... else |
| = += –= \*= ÷= %= ^= | Various assignments |
| &= |= <<= >>= >>>= |  |