Day-Five

Table of Contents

Notes

Links

1. RegExLib

Java

Operators and Expressions

```
package com.pluralsight;

public class BasicFloatingPointMathApp {

    public static void main(String[] args){
        float a = 10;
        float b = 3;
        float result;
        result = a + b;
        System.out.println(result); // displays 13.0
        result = a - b;
        System.out.println(result); // displays 7.0
        result = a * b;
```

```
        System.out.println(result); // displays 30.0
        result = a / b;
        System.out.println(result); // displays 3.3333333
        result = a % b;
        System.out.println(result); // displays 1.0
        result = b - (a % b);
        System.out.println(result); // displays 2.0
    }
}

/*
Parentheses can be placed around expressions to control

the order in which they are evaluated

Expressions can be a combination of integer and floating point

values, but if the result is a float point value, it must be assigned to a

floating point variable
*/
```

Pre/Post- Increment and Decrement

```
// Like other C-based languages, Java embraces the
// pre/post- increment and decrement operator

int x = 5;

int y;

x******; ******x; **y = ++x;** // adds 1 to x // adds 1 to x (now it is 6)

(now it is 7)

// adds 1 to x and then assigns x to y

// x is now 8 and y is 8

x = 5;

**y = x++;** // assigns x to y and then adds 1 to x
```

```
// x is now 6 and y is 5
```

Literals Are Typed

```
// examples

101 <- an int

101**L** <- a long (can use a little l, but it looks like a 1)

7.25 <- a double

7.25**f** <- a float

**"**Hi**"** <- a String

**'**a**'** <- a char

true <- a boolean

// Starting in Java SE 7 you can place underscore characters between digits
in a numerical literal

long ssn = 111_22_3333L;

long creditCard = 5200_7500_6500_0001L;

double loanBalance = 1_225_570.00;
```

When using floating-point literals in Java, you must decide between using 32-bit (`float`) or 64-bit (`double`) values:

1. **32-bit floating-point (`float`)**:
    - To indicate a float literal, append an "f" to the number (e.g., `7.12f`).
    - Example:

```
public class PriceApp {
    public static void main(String args[]){
```

```
        int count = 11;
        float unitPrice = 7.12f;
        float taxRate = 0.825f;
        float totalCost = (count * unitPrice) * (1 + taxRate);
        System.out.println(totalCost);
    }
}
```

Without the "f" suffix, there would be a syntax error.

2. **64-bit floating-point (`double`)**:
    - Double literals do not need a suffix.
    - Example:

```
public class PriceApp {
    public static void main(String args[]){
        int count = 11;
        double unitPrice = 7.12;
        double taxRate = 0.825;
        double totalCost = (count * unitPrice) * (1 + taxRate);
        System.out.println(totalCost);
    }
}
```

Handling Floating-Point Literals and Type Conversions in Java: 32-bit `float` vs. 64-bit `double`

Widening Conversions

- Widening occurs when a value is assigned to a variable of a larger data type, and is considered "safe" since no precision is lost. This occurs in the following order:
    - `byte -> short -> char -> int -> long -> float -> double`

Example:

```
int myInt = 9;
long myLong = myInt;       // Widening: int to long
```

```
float myFloat = 3.8f;
double myDouble = myFloat; // Widening: float to double
```

Narrowing Conversions

- Narrowing occurs when assigning a larger data type to a smaller one. This may lead to data loss and requires explicit casting:
    - `double -> float -> long -> int -> char -> short -> byte`

  Example:

```
long myLong = 9;
int myInt = (int) myLong;      // Explicit casting needed
double myDouble = 123.456789;
float myFloat = (float) myDouble;  // Narrowing with casting
```

Key Points:

- Widening is safe and automatic; narrowing requires explicit casting due to potential loss of precision.
  Type casting in Java is the process of converting one data type into another. There are two main types of type casting:

Understanding Type Casting in Java: Widening vs. Narrowing Conversions

1. Widening Casting (Automatic)

- This is the process of converting a smaller data type to a larger data type. It happens automatically without any explicit instruction from the programmer.
- It is also called **implicit casting**.

The data types in widening casting follow this hierarchy:

```
byte -> short -> char -> int -> long -> float -> double
```

Example:

```
int myInt = 100;
long myLong = myInt;  // Automatic casting from int to long
float myFloat = myLong;  // Automatic casting from long to float
double myDouble = myFloat;  // Automatic casting from float to double
```

In this process, there is no loss of precision for **integer** types, but converting from **float** to **double** could still preserve precision since `double` has more capacity.

2. Narrowing Casting (Manual)

- Narrowing casting involves converting a larger data type into a smaller data type. This cannot be done automatically and requires **explicit casting**.
- It is also called **explicit casting**, and there's a risk of data loss because the larger type might contain more information than the smaller type can hold.

The narrowing casting hierarchy is:

```
double -> float -> long -> int -> char -> short -> byte
```

Example:

```
double myDouble = 9.78;
float myFloat = (float) myDouble;  // Manual casting from double to float
long myLong = (long) myFloat;      // Manual casting from float to long
int myInt = (int) myLong;          // Manual casting from long to int
```

Here, casting from a larger type to a smaller type can result in the **loss of precision**. For instance, the fractional part of `myDouble` is lost when cast to `long` or `int`.

Important Considerations:

- **Widening** is safe and automatic because there's no risk of data loss.
- **Narrowing** requires explicit casting because it may lead to loss of data (e.g., truncating decimals when casting from `double` to `int`).

Example of narrowing that causes data loss:

```
double myDouble = 9.99;
int myInt = (int) myDouble;  // Output will be 9 (decimal part is lost)
System.out.println(myInt);
```

Java's Math Class

[Java's Math Class]

The `Math` class in Java is a part of the `java.lang` package, and it provides a set of methods and constants for performing basic mathematical operations such as exponentiation, logarithms, trigonometry, and more. It is a utility class, so all methods are `static`, meaning you do not need to create an object to use them.

Key Features of the Math Class:

1. **Mathematical Constants**:
   - `Math.PI`: Represents the value of π (approximately 3.14159).
   - `Math.E`: Represents the value of Euler's number e (approximately 2.71828).
2. **Basic Operations**:
   - `Math.abs(value)`: Returns the absolute value of a number.
   - `Math.max(value1, value2)`: Returns the larger of two values.
   - `Math.min(value1, value2)`: Returns the smaller of two values.
   - `Math.pow(base, exponent)`: Raises the base to the power of the exponent.
   - `Math.sqrt(value)`: Returns the square root of a number.
   - `Math.cbrt(value)`: Returns the cube root of a number.

   Example:

   ```
   int a = -5;
   int b = 3;
   int maxVal = Math.max(a, b);  // returns 3
   double squareRoot = Math.sqrt(16);  // returns 4.0
   ```

3. **Rounding Functions**:

- `Math.ceil(value)`: Rounds a number up to the nearest integer.
- `Math.floor(value)`: Rounds a number down to the nearest integer.
- `Math.round(value)`: Rounds a floating-point number to the nearest integer (up if the fractional part is 0.5 or greater, otherwise down).

Example:

```
double num = 7.25;
System.out.println(Math.ceil(num));  // returns 8.0
System.out.println(Math.floor(num)); // returns 7.0
System.out.println(Math.round(num)); // returns 7
```

4. **Trigonometric Functions**:

- `Math.sin(radians)`: Returns the sine of an angle (in radians).
- `Math.cos(radians)`: Returns the cosine of an angle (in radians).
- `Math.tan(radians)`: Returns the tangent of an angle (in radians).
- `Math.toRadians(degrees)`: Converts degrees to radians.
- `Math.toDegrees(radians)`: Converts radians to degrees.

Example:

```
double angleDegrees = 45.0;
double angleRadians = Math.toRadians(angleDegrees);
double sineValue = Math.sin(angleRadians);  // returns 0.7071
```

5. **Exponential and Logarithmic Functions**:

- `Math.exp(value)`: Returns the value of e raised to the power of the argument.
- `Math.log(value)`: Returns the natural logarithm (base e) of the argument.
- `Math.log10(value)`: Returns the base 10 logarithm of the argument.

Example:

```
double expVal = Math.exp(1);  // returns 2.71828
double logVal = Math.log(10); // returns 2.30258
```

6. **Random Number Generation**:

- `Math.random()`: Returns a random double between 0.0 (inclusive) and 1.0 (exclusive).

Example:

```
double randomValue = Math.random();  // returns a random value between 0
and 1
```

Summary:

The `Math` class is a powerful utility in Java that provides easy access to a wide range of mathematical functions. Whether you need basic arithmetic operations, trigonometric calculations, or random number generation, the `Math` class offers built-in methods to handle these tasks efficiently.

Exercise 2

```
package com.pluralsight;



/*
*

1. Create two variables to represent the salary for Bob and Gary, name them
bobSalary and garySalary. Create a new variable named
highestSalary. Determine whose salary is greater using Math.max() and
store the answer in highestSalary. Set the initial salary variables to any
value
you want. Print the answer (i.e "The highest salary is …")

2. Find and display the smallest of two variables named carPrice and
truckPrice. Set the variables to any value you want.

3. 4.
Find and display the area of a circle whose radius is 7.25
Find and display the square root a variable after it is set to 5.0

*
5. 6. 7.
Find and display the distance between the points (5, 10) and (85, 50)
Find and display the absolute (positive) value of a variable after it is set
to -3.8
```

```java
Find and display a random number between 0 and 1

*
* */


import java.util.Random;

public class MathApp {

    public static void main(String[] args) {


//        Question One
//        Declaration and Initialization
        double bobSalary = 10000.24 ;
        double garySalary = 20000;
        double highestSalary = Math.max(bobSalary, garySalary);

        System.out.println("The hightest salary is : " + highestSalary);


//        Question Two
//        Declaration and Initialization
        double carPrice = 45000.25;
        double truckPrice = 70000.50;
        double cheapestPrice = Math.min(carPrice, truckPrice);

        System.out.println("The cheapestPrice is: " + cheapestPrice);

//        Question Three
//        Declaration and Initialization
        double radius = 7.25;
        double area = Math.PI * Math.pow(radius, 2);

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

//        Question Four
//        Declaration and Initialization
        double sqrtRoot = 5;
        double sqrt = Math.sqrt(sqrtRoot);

        System.out.println("The squrt :" + sqrt);


//        Question Five
```

```java
//        Declaration and Initialization
        int x1 = 5, y1 = 10;
        int x2 = 85, y2 = 50;
        double distance = Math.sqrt(Math.pow((x2 - x1), 2) + Math.pow((y2 -
y1), 2));

        System.out.println("The distance between the points (5, 10) and (85,
50) is: " + distance);

//        Question Six
//        Declaration and Initialization
        double value = -3.8;
        double absoluteValue = Math.abs(value);
        System.out.println("Absolute value of -3.8: " + absoluteValue);

//        Question Seven
//        Declaration and Initialization
        double randomNumber = new Random().nextDouble(0,1);
        System.out.println("The Random Number (between 0 - 1): " +
randomNumber);

    }
}
```

```
The hightest salary is : 20000.0
The cheapestPrice is: 45000.25
The area of the cirlce: 165.1299638543135
The squrt :2.23606797749979
The distance between the points (5, 10) and (85, 50) is: 89.44271909999159
Absolute value of -3.8: 3.8
The Random Number (between 0 - 1): 0.3296389257638569

Process finished with exit code 0
```

Different Ways to Printout in Java, String.format(), Scanner

Printout

1. Using `System.out.println()`

This method is the most straightforward way to print data to the console.

```
System.out.println("Hello, World!");
```

2. Using `System.out.print()`

Similar to `println()`, but it does not append a newline character at the end.

```
System.out.print("Hello, ");
System.out.print("World!");
```

3. Using `System.out.printf()`

This method is used for formatted output. You can use format specifiers to control the output format.

```
int number = 10;
String text = "apples";
System.out.printf("I have %d %s.%n", number, text);
```

String.format()

The `String.format()` method in Java is used to create formatted strings, allowing you to insert values into a string template in a structured way. It is similar to the `printf` function in C or other languages. This method is particularly useful when you want to control the format of output, such as number of decimal places, alignment, padding, and so on.

Syntax

```
String formattedString = String.format(String format, Object... args);
```

- **format**: A format string that specifies how to format the output.
- **args**: The arguments that will be formatted and inserted into the format string.

Format Specifiers

The format string can include format specifiers that start with `%`, followed by optional flags, width, precision, and a conversion character. Here are some common format specifiers:

Specifier	Description	Example
`%d`	Decimal integer	`String.format("%d", 42);` // Output: "42"`
`%f`	Floating-point number	`String.format("%.2f", 3.14159);` // Output: "3.14"`
`%s`	String	`String.format("Hello, %s!", "World");` // Output: "Hello, World!"`
`%c`	Character	`String.format("%c", 'A');` // Output: "A"`
`%x`	Hexadecimal integer	`String.format("%x", 255);` // Output: "ff"`
`%b`	Boolean value	`String.format("%b", true);` // Output: "true"`
`%n`	Platform-independent newline	`String.format("Hello%nWorld");` // Output: "Hello\nWorld"`

Example Usage

1. Basic Formatting

```
int age = 30;
String name = "Alice";
String formattedString = String.format("%s is %d years old.", name, age);
System.out.println(formattedString); // Output: "Alice is 30 years old."
```

2. Floating-point Formatting

```
double price = 12.34567;
String formattedPrice = String.format("The price is $%.2f.", price);
System.out.println(formattedPrice); // Output: "The price is $12.35."
```

3. Padding and Alignment

You can use width specifiers to pad or align text.

```
String formatted = String.format("|%10s|%-10s|%5d|", "right", "left", 42);
```

```
System.out.println(formatted); // Output: "|     right|left      |   42|"
```

- `%10s` means to right-align the string in a field of width 10.
- `%-10s` means to left-align the string in a field of width 10.
- `%5d` means to right-align the integer in a field of width 5.

4. Using Multiple Specifiers

```
String firstName = "John";
String lastName = "Doe";
int score = 95;

String result = String.format("Student: %s %s, Score: %d", firstName,
lastName, score);
System.out.println(result); // Output: "Student: John Doe, Score: 95"
```

Scanner

Consuming a Line Separator

Example

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

System.out.print("Enter number 1: ");
int num1 = scanner.nextInt();

System.out.print("Enter number 2: ");
int num2 = scanner.nextInt();
scanner.nextLine();   // "eat" the leftover CRLF

System.out.println("What do you want to do? ");
System.out.print("   Enter 'add' or 'subtract': ");
String action = scanner.nextLine();

System.out.println("Preparing to do math... ");
```

TRANSCRIPT OF RUNNING CODE:

```
Enter number 1: 13↵
Enter number 2: 145↵
What do you want to do?
   Enter 'add' or 'subtract': add↵
Preparing to do math...
```

If you use `nextInt()` before calling `nextLine()`, you might encounter an issue where the `nextLine()` call immediately returns an empty string without giving you a chance to input anything. This happens because `nextInt()` only consumes the integer part of the input, leaving the newline character (`\n`) in the input buffer. When `nextLine()` is called, it reads that leftover newline character instead of waiting for user input.

Here's an example of the problem:

```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a number: ");
```

```
        int number = scanner.nextInt();  // Reads the integer but leaves the
newline

        System.out.print("Enter a line of text: ");
        String input = scanner.nextLine();  // Immediately reads the
leftover newline

        System.out.println("You entered number: " + number);
        System.out.println("You entered text: " + input);  // input will be
an empty string
    }
}
```

How to fix it:

You can add an extra `scanner.nextLine()` after `nextInt()` to consume the leftover newline character before calling `nextLine()` for actual input:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a number: ");
        int number = scanner.nextInt();
        scanner.nextLine();  // Consumes the leftover newline

        System.out.print("Enter a line of text: ");
        String input = scanner.nextLine();  // Reads the actual line

        System.out.println("You entered number: " + number);
        System.out.println("You entered text: " + input);
    }
}
```

In this version, the extra `scanner.nextLine()` consumes the newline character left by `nextInt()`, allowing the following `nextLine()` to work as expected.

Scanner Class

Method	Description	Example Usage
`next()`	Reads the next token (word) from the input.	`String token = scanner.next();`
`nextLine()`	Reads an entire line of input, including spaces.	`String line = scanner.nextLine();`
`nextInt()`	Reads the next integer from the input.	`int number = scanner.nextInt();`
`nextDouble()`	Reads the next double from the input.	`double value = scanner.nextDouble();`
`nextBoolean()`	Reads the next boolean value from the input.	`boolean flag = scanner.nextBoolean();`
`hasNext()`	Returns true if there is another token in the input.	`if (scanner.hasNext()) { ... }`
`hasNextInt()`	Returns true if the next token can be interpreted as an integer.	`if (scanner.hasNextInt()) { ... }`
`hasNextDouble()`	Returns true if the next token can be interpreted as a double.	`if (scanner.hasNextDouble()) { ... }`
`close()`	Closes the scanner and releases any associated resources.	`scanner.close();`

Example Usage of the Scanner Class

Here's a complete example demonstrating the use of the `Scanner` class to gather different types of user input:

```
import java.util.Scanner;

public class ScannerExample {
    public static void main(String[] args) {
        // Create a Scanner object to read input from the console
        Scanner scanner = new Scanner(System.in);

        // Prompt the user for their name and read it
        System.out.print("Enter your name: ");
        String name = scanner.nextLine(); // Reads an entire line

        // Prompt the user for their age and read it
        System.out.print("Enter your age: ");
```

```
        int age = scanner.nextInt(); // Reads an integer

        // Prompt the user for their height and read it
        System.out.print("Enter your height in meters: ");
        double height = scanner.nextDouble(); // Reads a double

        // Prompt the user for their student status and read it
        System.out.print("Are you a student? (true/false): ");
        boolean isStudent = scanner.nextBoolean(); // Reads a boolean

        // Output the collected data
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Height: " + height);
        System.out.println("Student: " + isStudent);

        // Close the scanner
        scanner.close();
    }
}
```

Explanation of Example

1. **Scanner Creation**: A `Scanner` object is created to read input from the console.
2. **Input Prompts**: The program prompts the user to enter their name, age, height, and student status.
3. **Reading Input**: Different `Scanner` methods are used to read different types of input:
   - `nextLine()` for reading a full line (name).
   - `nextInt()` for reading an integer (age).
   - `nextDouble()` for reading a double (height).
   - `nextBoolean()` for reading a boolean (student status).
4. **Output**: The collected data is printed to the console.
5. **Closing the Scanner**: The scanner is closed at the end of the program to free up resources.

Exercise 3

```
package com.pluralsight;
```

```java
import java.util.Scanner;

public class BasicCalculator {

    public static void main(String[] args) {


        for (int i = 0; i < 4; i++) {
            calculator();
        }



    }

    public static void calculator() {


        // Create a Scanner object
        Scanner scanner = new Scanner(System.in);
        System.out.println("Hello and welcome!");
        System.out.println("This is a basic calculator program.");
        System.out.println("It will add two numbers and print the result.");

        // Get the first number
        System.out.println("Enter the first number:");
        int firstNumber = scanner.nextInt();
        // Get the second number
        System.out.println("Enter the second number:");
        int secondNumber = scanner.nextInt();

        // Get the operation
        System.out.println("Possible calculations:\n" +
                "(A)dd\n" +
                "(S)ubtract\n" +
                "(M)ultiply\n" +
                "(D)ivide\n" +
                "Please select an option:"
        );
        char operation = scanner.next().charAt(0);
        double result = 0;

        // Perform the operation
        if (operation == 'A' || operation == 'a') {
            result = firstNumber + secondNumber;
        } else if (operation == 'S' || operation == 's') {
```

```
            result = firstNumber - secondNumber;
        } else if (operation == 'M' || operation == 'm') {
            result = firstNumber * secondNumber;
        } else if (operation == 'D' || operation == 'd') {
            if (secondNumber == 0) {  // Check if the second number is 0
                System.out.println("Your second number cannot be 0");
                return;
            }
            result = firstNumber / secondNumber;
        } else {
            System.out.println("Invalid operation. Please try again.");
            return;
        }

        // Print the result
        System.out.println("Result : " + result);

    }



}
```