Module II: Selection Statements, Loops, Characters and Strings

Course: Object oriented Programming in JAVA

Dr. Vishwa Pratap Singh

NextGen Academy

February 10, 2024

NextGen Academy

Towards fulfilling a million dreams

NextGen Academy

Towards fulfilling a million dreams

Boolean Data Type

- Used for storing true/false values.
- Takes 1 byte of space.
- True is represented as 1, false as 0.
- Commonly used in decision statements.
- Derived from Boolean algebra.

If Statements in Java

- Used for conditional execution.
- Basic syntax: `if (condition) { /* code */ }`.
- Example: Checking if age is greater than or equal to 18.
- Optional `else` block for false conditions.
- Essential for implementing conditional logic.

- Demonstrated another example with `if` condition.
- Explained the process with code comments.
- Highlighted the output based on the condition.

Two-Way if-else Statements

- Used to choose between two possibilities.
- Syntax: `if (condition) { /* code */ } else { /* code */ }`.
- Flowchart representation of execution.

- Executing different blocks based on conditions.
- Introduction to nested control structure.
- Example of multiway selection.
- Demonstrated the flowchart for nested if-else.

Nested if - Example

- Discussed the possibility of embedding if statements.
- Used an example with a variable `num`.
- Code for writing English word for `num` value.

Multiway Selection in Java

- Standard multiway selection structure.
- Used when selecting one option from several.
- Illustrated with a flowchart.
- Example of code for writing English words.

Usage of if-else Structure

- Explained the use of if-else structure in programming.
- Compared two versions of the code for clarity.
- Highlighted the decision-making process.

Logical Operators Overview

- Logical operators perform logical *AND*, *OR*, and *NOT* operations.
- Used to combine conditions or complement evaluations in decision-making.
- Short-circuiting effect in *AND* and *OR* operators.

AND Operator (&&)

- Returns true when both conditions are true.
- Syntax: `condition1 && condition2`
- Example code in Java.
- Short-circuiting effect demonstrated.

```
// Java code snippet
if ((a < b) && (b == c)) {
    d = a + b + c;
    System.out.println("The sum is: " + d);
}
```

NextGen Academy
Towards fulfilling a million dreams

OR Operator (——)

- Returns true when at least one condition is true.
- Syntax: `condition1` — condition2—
- Example code in Java.
- Short-circuiting effect demonstrated.

```
// Java code snippet
if (a > b || c == d) {
    System.out.println("One or both conditions are true");
} else {
    System.out.println("Both conditions are false");
}
```

NextGen Academy
Towards fulfilling a million dreams

NOT Operator (!)

- Unary operator, returns true when the condition is false.
- Syntax: `!(condition)`
- Example code in Java.

```
// Java code snippet
System.out.println("!(a < b) = " + !(a < b));
System.out.println("!(a > b) = " + !(a > b));
```

Implementing Logical Operators

- Java program demonstrating logical operators on boolean values.
- Truth table for *AND*, *OR*, and *NOT*.

```
// Java program snippet
boolean a = true;
boolean b = false;

System.out.println("a && b: " + (a && b));
System.out.println("a || b: " + (a || b));
System.out.println("!a: " + !a);
System.out.println("!b: " + !b);
```

NextGen Academy
Towards fulfilling a million dreams

Advantages of Logical Operators

1. Readability: Logical operators enhance code readability.
2. Flexibility: They allow flexible combination for complex conditions.
3. Reusability: Logical operators enable code reuse in different parts of a program.
4. Debugging: Simplifies debugging process by isolating conditions.

Disadvantages of Logical Operators

1. Short-circuit evaluation: Can lead to subtle bugs if not used carefully.
2. Limited expressiveness: Less powerful than if-else and switch-case constructs.
3. Potential for confusion: Parentheses are often needed to clarify order of operations.
4. Boolean coercion: Unexpected behavior with non-Boolean values.

- Java Switch Statements
- Mathematical Functions in Java

Switch Statements Overview

- Used for selective or multiple-branch decision-making.
- Basic syntax:

```
switch (expression) {
  case value1:
    // Code if expression == value1
    break;
  // More cases and default
}
```

- The switch expression is evaluated once, and its value is compared to each case value.
- If a case value matches the switch expression, the corresponding code block is executed.
- The break statement is used to exit the switch statement after a case is matched. If omitted, execution will "fall through" to the next case.
- The default case is optional and executed if none of the case values match the switch expression.

Switch Statements Example

```
public class SwitchExample {
  public static void main(String[] args) {
    int dayOfWeek = 3;
    switch (dayOfWeek) {
      case 1:
        System.out.println("Sunday");
        break;
      case 2:
        System.out.println("Monday");
        break;
      // More cases
      default:
        System.out.println("Invalid day");
    }
  }
```

NextGen Academy
Towards fulfilling a million dreams

Mathematical Functions Overview

- Predefined mathematical functions in Java.
- Using the Math class.

abs() Function

- Returns the absolute value of a number.
- Syntax:

```
int abs(int num);
long abs(long num);
float abs(float num);
double abs(double num);
```

- Example Java program to input an integer and print its absolute value.

```
import java.util.Scanner;
public class Example {
  public static void main(String args[]) {
    int n, x;
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter an integer number ");
    n = sc.nextInt();
    x = Math.abs(n);
    System.out.println("Absolute value of " + n + " is "
  }
```

min() Function

- Returns the smallest among two numbers.
- Syntax:

```
int min(int a, int b);
long min(long a, long b);
float min(float a, float b);
double min(double a, double b);
```

- Example Java program to input two integers and print the smallest value on the screen.

```
import java.util.Scanner;
public class Example {
  public static void main(String args[]) {
    int a, b, c;
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter two integer numbers");
    a = sc.nextInt();
    b = sc.nextInt();
    c = Math.min(a, b);
```

max() Function

- Returns the greater among two numbers.
- Syntax:

```
int max(int a, int b);
long max(long a, long b);
float max(float a, float b);
double max(double a, double b);
```

- Example Java program to input two integers and print the greater value on the screen.

```
import java.util.Scanner;
public class Example {
  public static void main(String args[]) {
    int a, b, c;
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter two integer numbers");
    a = sc.nextInt();
    b = sc.nextInt();
    c = Math.max(a, b);
```

sqrt() Function

- Returns the square root of a positive number.
- Syntax:

```
double sqrt(double num);
```

- Example Java program to input an integer and print its square root.

```
import java.util.Scanner;
public class Example {
  public static void main(String args[]) {
    int n;
    double x;
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter an integer number ");
    n = sc.nextInt();
    x = Math.sqrt(n);
    System.out.println("Square root of " + n + " is " +
  }
}
```

NextGen Academy
Towards fulfilling a million dreams

- Output: Enter an integer number 25 Square root of 25 is 5.0

ceil() Function

- Returns the nearest integer greater than the number passed as an argument.
- Syntax:

```
double ceil(double num);
```

- Example Java program to input a floating-point number and print its ceil value.

```
import java.util.Scanner;
public class Example {
  public static void main(String args[]) {
    float n;
    double x;
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a floating-point (decimal) n
    n = sc.nextFloat();
    x = Math.ceil(n);
    System.out.println("Ceil value of " + n + " is " + x
  }
}
```

NextGen Academy
Towards fulfilling a million dreams

- Output: Enter a floating-point (decimal) number 12.4 Ceil value of 12.4 is 13.0

floor() Function

- Returns the nearest integer number less than the number passed as an argument.
- Syntax:

```
double floor(double num);
```

- Example Java program to input a floating-point number and print its floor value.

```
import java.util.Scanner;
public class Example {
  public static void main(String args[]) {
    float n;
    double x;
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a floating-point (decimal) n
    n = sc.nextFloat();
    x = Math.floor(n);
    System.out.println("Floor value of " + n + " is " +
  }
```

pow() Function

- Computes the power of a number.
- Syntax:

```
double pow(double x, double y);
```

- Example Java program to input two integer numbers and print the power.

```
import java.util.Scanner;
public class Example {
  public static void main(String args[]) {
    int a, b;
    double c;
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter two integer numbers");
    a = sc.nextInt();
    b = sc.nextInt();
    c = Math.pow(a, b);
    System.out.println("Power = " + c);
  }
}
```

Other Mathematical Functions

- List of other useful mathematical functions:
  - round, rint, cos, acos, cosh, sin, asin, sinh, tan, atan, atan2, tanh, exp, log, log10.
- Explanation of each function.

round() Function

- Returns the closest round-up value of a given number.
- Syntax:

```
int round(float x);
long round(double x);
```

- Example Java program to round a floating-point number.

```
import java.util.Scanner;
public class Example {
  public static void main(String args[]) {
    float n;
    int roundedValue;
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a floating-point (decimal) n
    n = sc.nextFloat();
    roundedValue = Math.round(n);
    System.out.println("Rounded value of " + n + " is "
  }
}
```

NextGen Academy
Towards fulfilling a million dreams

rint() Function

- Returns the double value that is closest in value to the argument and is equal to a mathematical integer.
- Syntax:

```
double rint(double x);
```

- Example Java program to demonstrate rint function.

```
public class Example {
  public static void main(String args[]) {
    double n, roundedValue;
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a floating-point number ");
    n = sc.nextDouble();
    roundedValue = Math.rint(n);
    System.out.println("Rounded value of " + n + " is "
  }
}
```

- Output: Enter a floating-point number 12.8, Rounded value of 12.8 is 13.0.

cos() Function

- Returns the cosine of an angle in radians.
- Syntax:

```
double cos(double x);
```

- Example Java program to calculate the cosine of an angle.

```
public class Example {
  public static void main(String args[]) {
    double angle, cosineValue;
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter an angle in radians ");
    angle = sc.nextDouble();
    cosineValue = Math.cos(angle);
    System.out.println("Cosine of " + angle + " radians
  }
}
```

- Output: Enter an angle in radians 1.2, Cosine of 1.2 radians is 0.3623577544766736.

acos() Function

- Returns the arc cosine of a value, which will be in the range [0, pi].
- Syntax:

```
double acos(double x);
```

- Example Java program to calculate the arc cosine of a value.

```
public class Example {
  public static void main(String args[]) {
    double value, arcCosine;
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a value between -1 and 1 ");
    value = sc.nextDouble();
    arcCosine = Math.acos(value);
    System.out.println("Arc cosine of " + value + " is "
  }
}
```

- Output: Enter a value between -1 and 1 0.5, Arc cosine of 0.5 is 1.0471975511965979.

- Returns the hyperbolic cosine of a value.
- Syntax:

```
double cosh(double x);
```

- Example Java program to calculate the hyperbolic cosine of a value.

```
public class Example {
  public static void main(String args[]) {
    double value, hyperbolicCosine;
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter■a■value■");
    value = sc.nextDouble();
    hyperbolicCosine = Math.cosh(value);
    System.out.println("Hyperbolic■cosine■of■" + value
  }
}
```


- Output: Enter a value 2.0, Hyperbolic cosine of 2.0 is 3.7621956910836314.

Additional Mathematical Functions

- Remaining mathematical functions:
  - sin, asin, sinh, tan, atan, atan2, tanh, exp, log, log10.
- Explanation of each function.

Character Data Type

- The `char` data type represents symbols such as alphabets and numbers.
- Size: 16-bit, range: 0 to 65,535, ASCII range: 0 to 127.
- Syntax: `char variable_name = 'variable_value';`
- Characteristics:
  - Range: 0 to 65,535.
  - Default value: '\u0000' (lowest Unicode range).
  - Size: 2 bytes (Java uses Unicode).

Displaying Characters

```
public class example {
  public static void main(String[] args) {
    char c1 = 'x';
    char c2 = 'X';
    System.out.println("c1 is: " + c1);
    System.out.println("c2 is: " + c2);
  }
}
```

NextGen Academy
Towards fulfilling a million dreams

Printing Characters Using ASCII Value

```
public class example {
  public static void main(String[] args) {
    char c1, c2, c3;
    c1 = 65;
    c2 = 'B';
    c3 = 67;
    System.out.println("The■characters■are:■" + c1 + c2 + c3);
  }
}
```

NextGen Academy
Towards fulfilling a million dreams

Incrementing and Decrementing the chars

```
public class example {
  public static void main(String[] args) {
    char c1 = 'A';
    System.out.println("The value of c1 is: " + c1);
    c1++;
    System.out.println("After incrementing: " + c1);
    c1--;
    System.out.println("After decrementing: " + c1);
  }
}
```

NextGen Academy
Towards fulfilling a million dreams

Breaking String Into Character Java

```
import java.util.Arrays;

public class example {
  public static void main(String[] args) {
    String str1 = "Saket";
    char[] chars = str1.toCharArray();
    System.out.println("Original String was: " + str1);
    System.out.println("Characters are: " + Arrays.toString(cha
  }
}
```

NextGen Academy
Towards fulfilling a million dreams

Represent `char` Into Unicode System

```
import java.util.Arrays;

public class example {
  public static void main(String[] args) {
    char chars1 = '\u0058';
    char chars2 = '\u0059';
    char chars3 = '\u005A';
    System.out.println("chars1 , chars2 , and chars2 are: " + cha
  }
}
```

NextGen Academy
Towards fulfilling a million dreams

Typecast Integer To `char` Java

```
import java.util.Arrays;

public class example {
  public static void main(String[] args) {
    int number1 = 66;
    char chars1 = (char)number1;
    // Similar code for other variables...
    System.out.println(chars1);
    // Print other variables...
  }
}
```

NextGen Academy
Towards fulfilling a million dreams

- Java's string is an object representing a sequence of characters.
- An array of characters works the same as Java string.
- Example: `char[]`
  `ch={'j','a','v','a','t','p','o','i','n','t'};`
- String methods: `compare()`, `concat()`, `equals()`, `split()`, `length()`, `replace()`, `compareTo()`, `intern()`, `substring()`.

CharSequence Interface

- `CharSequence` interface represents a sequence of characters.
- Implemented by `String`, `StringBuffer`, and `StringBuilder`.
- `String` is immutable; `StringBuffer` and `StringBuilder` are mutable.
- `String` implements `Serializable`, `Comparable`, and `CharSequence` interfaces.

What is String in Java?

- String is an object representing a sequence of characters.
- `java.lang.String` class is used to create a string object.

How to Create a String Object?

1. By string literal.
2. By `new` keyword.

String Literal

- Created using double quotes, e.g., `String s="welcome";`.
- JVM checks ”string constant pool” first.
- If the string already exists, a reference is returned; otherwise, a new instance is created.

```
String s1 = ”Welcome”;
String s2 = ”Welcome”; // Returns reference to s1
```

By `new` Keyword

```
String s = new String("Welcome");
```

- Creates two objects and one reference variable.
- String object in heap memory, literal in the string constant pool.
- `s` refers to the object in the heap.

NextGen Academy
Towards fulfilling a million dreams

Java String Example

```
public class StringExample{
  public static void main(String args[]){
    String s1="java";
    char ch[]={'s','t','r','i','n','g','s'};
    String s2=new String(ch);
    String s3=new String("example");
    System.out.println(s1);
    System.out.println(s2);
    System.out.println(s3);
  }
}
```

Java String Class Methods

1. `charAt(int index)`
2. `length()`
3. `format(String format, Object... args)`
4. `format(Locale l, String format, Object... args)`
5. `substring(int beginIndex)`
6. `substring(int beginIndex, int endIndex)`
7. `contains(CharSequence s)`
8. `join(CharSequence delimiter, CharSequence... elements)`
9. `join(CharSequence delimiter, Iterable<? extends CharSequence> elements)`
10. `equals(Object another)`

NextGen Academy
Towards fulfilling a million dreams

More Java String Class Methods

11. `isEmpty()`
12. `concat(String str)`
13. `replace(char old, char new)`
14. `replace(CharSequence old, CharSequence new)`
15. `equalsIgnoreCase(String another)`
16. `split(String regex)`
17. `split(String regex, int limit)`
18. `intern()`
19. `indexOf(int ch)`
20. `indexOf(int ch, int fromIndex)`

NextGen Academy
Towards fulfilling a million dreams

More Java String Class Methods (contd.)

```
// Example: Using some methods
String exampleString = "Hello, World!";
char charAtIndex = exampleString.charAt(7);
int length = exampleString.length();
String substring = exampleString.substring(7, 12);
boolean contains = exampleString.contains("World");
String[] splitArray = exampleString.split(",");
// Add more examples as needed
```

NextGen Academy
Towards fulfilling a million dreams

The for Loop

- Used for repetitive execution with a fixed number of iterations.
- Provides a concise way for loop initialization, condition checking, and variable incrementing or decrementing.
- Basic syntax: `for (initialization; condition; update) { // Code }`
- Example: `for (int i = 1; i <= 5; i++) { System.out.println(i); }`
- Flow chart illustration.
- Common use cases: iterating over arrays, lists, or performing a specific task a certain number of times.

NextGen Academy
Towards fulfilling a million dreams

For Loop Example

- Demonstrates counting from 1 to 5.
- Code: `for (int i = 1; i <= 5; i++) { System.out.println(i); }`
- Output: `1 2 3 4 5`
- Explanation:
  - The loop initializes `i` to 1.
  - Checks if `i` is less than or equal to 5.
  - Increments `i` by 1 in each iteration.
  - Prints the value of `i`.

For Loop with Arrays

- Can iterate over arrays or iterable collections.
- Example: `for (int i = 0; i < numbers.length; i++) { System.out.println(numbers[i]); }`
- Demonstrates iterating over an array of integers.
- Use case: accessing and processing each element of an array.

For Loop Programs

- Program: Counting from 1 to 5.
- Code: `public class ForLoopDemo { ... }`
- Output: `1 2 3 4 5`
- Explanation:
  - The loop initializes `i` to 1, checks if `i` is less than or equal to 5, and increments `i` by 1.
  - Prints the value of `i`.

While Loop

- Allows repetitive execution based on a specified condition.
- Basic syntax: `while (condition) { // Code }`
- Example: `int i = 1; while (i <= 5) { System.out.println(i); i++; }`
- Common use cases: when the number of iterations is not known beforehand or based on a dynamic condition.

While Loop Program

- Program: Calculate the sum of numbers from 1 to 10.
- Code: `public class WhileLoopDemo { ... }`
- Output: `The sum of numbers from 1 to 10 is: 55`
- Explanation:
  - Initializes a sum variable to store the sum of numbers.
  - Uses a while loop to calculate the sum, adding the current value of `i` to the sum in each iteration.

Do-While Loop

- Similar to the while loop but guarantees at least one execution.
- Basic syntax: `do { // Code } while (condition);`
- Example: Password input validation.
- Use case: ensuring that a certain block of code executes at least once.

Do-While Loop Program

- Program: Validate user password input.
- Code: `public class DoWhileLoopDemo { ... }`
- Explanation:
  - Uses a do-while loop to keep asking the user for a password until the correct password is entered.
  - The loop continues as long as the condition `!password.equals("secret")` is true.

Nested Loops

- Allows one or more loops inside another loop.
- Useful for complex iterations like 2D arrays or pattern generation.
- Example 1: Nested for loop for a 2D array.
- Example 2: Nested for loops to generate patterns.
- Use case: solving problems that involve multiple levels of iteration.

Nested Loops Example

- Example 1: Nested for loop for a 2D array.
- Code: `for (int i = 0; i < matrix.length; i++) { for (int j = 0; j < matrix[i].length; j++) { System.out.print(matrix[i][j] + " "); } System.out.println(); }`
- Output: `1 2 3`
  `4 5 6`
  `7 8 9`
- Example 2: Nested for loops to generate patterns.
- Code: `for (int i = 1; i <= n; i++) { for (int j = 1; j <= i; j++) { System.out.print("* "); } System.out.println(); }`
- Output: `*`
  `* *`
  `* * *`
  `* * * *`
  `* * * * *`

NextGen Academy
Towards fulfilling a million dreams

Break Statement

- The `break` statement is used to prematurely exit a loop or switch statement.
- Basic syntax: `break;`
- Commonly used within `for`, `while`, and `do-while` loops.

Example 1: Break in for loop

```
for (int i = 1; i <= 5; i++) {
    if (i == 3) {
        break; // Exit the loop when i is equal to 3
    }
    System.out.println(i);
}
```

- Output: 1, 2

Example 2: Break in while loop

```
int number = 1;
while (number <= 5) {
    if (number == 3) {
        break; // Exit the loop when number is equal to 3
    }
    System.out.println(number);
    number++;
}
```

- Output: 1, 2

Example 3: Using break in a switch statement

```
int dayOfWeek = 3;
switch (dayOfWeek) {
  case 1:
    System.out.println("Monday");
    break;
  case 2:
    System.out.println("Tuesday");
    break;
  case 3:
    System.out.println("Wednesday");
    break;
  default:
    System.out.println("Other day");
}
```

- Output: Wednesday

NextGen Academy
Towards fulfilling a million dreams

Continue Statement

- The `continue` statement skips the current iteration and moves to the next iteration of a loop.
- Used in `for`, `while`, and `do-while` loops.

Example 1: Using continue in a for loop

```
for (int i = 1; i <= 5; i++) {
    if (i == 3) {
        continue; // Skip the current iteration when i is equal to
    }
    System.out.println(i);
}
```

- Output: 1, 2, 4, 5

Example 2: Using continue in a while loop

```
int number = 1;
while (number <= 5) {
   if (number == 3) {
      number++;
      continue; // Skip the current iteration when number is equa
   }
   System.out.println(number);
   number++;
}
```

- Output: 1, 2, 4, 5

Example 3: Using continue in a do-while loop

```
int i = 1;
do {
    if (i \% 2 == 0) {
        i++;
        continue; // Skip even numbers
    }
    System.out.println(i);
    i++;
} while (i <= 5);
```

- Output: 1, 3, 5

NextGen Academy
Towards fulfilling a million dreams