Module 2: Looping Constructs and Arrays

Premanand S

Assistant Professor
School of Electronics Engineering (SENSE)
Vellore Instittute of Technology
Chennai Campus

premanand.s@vit.ac.in

February 17, 2025

Strings

Question 1:

- What is the difference between == and .equals() when comparing two strings? Provide an example

Answer 1:

- The == operator checks for reference equality, meaning it determines whether two references point to the same object in memory.
- The .equals() method checks for value equality, meaning it determines whether the content (characters) of the two strings is the same.

Answer 1:

Example (Understanding)

```
public class StringComparison {
    public static void main(String[] args) {
        String str1 = "Java";
        String str2 = "Java";
        String str3 = new String("Java");
        // New object in heap memory

        System.out.println(str1 == str2);
        System.out.println(str1 == str3);

        System.out.println(str1.equals(str2));
        // true (content is the same)
        System.out.println(str1.equals(str3));
        // true (content is the same)
    }}
```

Question 2:

- Write a program to count the number of characters in a given string, excluding spaces

Answer 2:

Example (Understanding)

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String input = scanner.nextLine();
        int count = 0;
        for (int i = 0; i < input.length(); i++) {
            char ch = input.charAt(i);
            if (ch != ’ ’) {
                count++;
            }        }
        System.out.println("Number of characters
        (excluding spaces): " + count);
        scanner.close(); }}
```

Question 3:

- Why does the following code not modify the original string?

Example

```
String str = "Java";
str.concat(" Programming");
System.out.println(str); // Output: Java
```

Answer 3:

- The reason the original string str is not modified in the code is due to the immutability of strings in Java
- Any operation that appears to modify a string (e.g., concat(), replace(), etc.) actually creates and returns a new string object instead of modifying the original one.

Example (Understanding)

```
String str = "Java";
str = str.concat(" Programming");
System.out.println(str);

StringBuilder sb = new StringBuilder("Java");
sb.append(" Programming");
System.out.println(sb.toString());
```

Question 4:

- Write a program to check if a given string is a palindrome (reads the same backward as forward).

Answer 4:

Example (Understanding)

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String input = scanner.nextLine();
        String normalized = input.replaceAll
        ("[^a-zA-Z0-9]", "").toLowerCase();
        String reversed = new StringBuilder(normalized)
        .reverse().toString();
        if (normalized.equals(reversed)) {
            System.out.println("The string is a palindrome.");
        } else {
            System.out.println("The string is not a palindrome
        }         scanner.close(); }}
```

Question 5:

- Write a program to find the frequency of each character in a string.

Answer 5:

Example (Understanding)

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String input = scanner.nextLine().toLowerCase();
        int[] frequency = new int[26];
        for (char ch : input.toCharArray()) {
            if (ch >= ’a’ && ch <= ’z’) {
                frequency[ch - ’a’]++; } }
```

Answer 5:

Example (Understanding)

```
System.out.println("Character Frequencies:");
for (int i = 0; i < 26; i++) {
    if (frequency[i] > 0) {
        System.out.println("’" + (char) (’a’ + i)
        + "’ : " + frequency[I]);
        }
        }
scanner.close();
}
}
```

Question 6:

- How would you find all occurrences of a substring in a given string?

Answer 6:

Example (Understanding)

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the main string: ");
        String mainString = scanner.nextLine();
        System.out.print("Enter the substring to find: ");
        String subString = scanner.nextLine();
        List<Integer> indices = findAllOccurrences
        (mainString, subString);
        if (indices.isEmpty()) {
            System.out.println("The substring \"" + subString
            + "\" was not found.");
```

Answer 6:

Example (Understanding)

```
    } else {
        System.out.println("The substring \"" + subString
        + "\" was found at indices: " + indices);
    }
    scanner.close();
}
public static List<Integer>
findAllOccurrences(String mainString, String subString) {
    List<Integer> indices = new ArrayList<>();
    int index = 0;
    while ((index = mainString.indexOf(subString, index))
    != -1) {
        indices.add(index);
        index += subString.length();
    }         return indices;}}
```

Question 7:

- Write a program to find the length of the longest substring without repeating characters

Answer 7:

Example (Understanding)

```
import java.util.HashMap;
import java.util.Scanner;

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

        System.out.print("Enter a string: ");
        String input = scanner.nextLine();

        int maxLength = findLongestSubstringLength(input);

        System.out.println("The length of the longest substri

        scanner.close();
    }
```

Answer 7:

Example (Understanding)

```
    public static int findLongestSubstringLength(String s) {
        HashMap<Character, Integer> charIndexMap = new HashMap
        int maxLength = 0;
        int start = 0;
        for (int end = 0; end < s.length(); end++) {
            char currentChar = s.charAt(end);
            if (charIndexMap.containsKey(currentChar)) {
                start = Math.max(start,
                charIndexMap.get(currentChar) + 1);
            }
            charIndexMap.put(currentChar, end);
            maxLength = Math.max(maxLength, end - start + 1);
        }
        return maxLength;    }
}
```

Question 8:

- Write a program to check if two strings are anagrams (contain the same characters in a different order).

Answer 8:

Example (Understanding)

```
import java.util.Arrays;
import java.util.Scanner;
public class Main {
    // Method to check if two strings are anagrams
    public static boolean areAnagrams(String str1,
    String str2) {
        // Remove spaces and convert to lowercase for uniformi
        str1 = str1.replaceAll("\\s", "").toLowerCase();
        str2 = str2.replaceAll("\\s", "").toLowerCase();
        // If lengths are different, they cannot be anagrams
        if (str1.length() != str2.length()) {
            return false;
        }
```

Answer 8:

Example (Understanding)

```
        // Convert to character arrays and sort
        char[] charArray1 = str1.toCharArray();
        char[] charArray2 = str2.toCharArray();

        Arrays.sort(charArray1);
        Arrays.sort(charArray2);

        // Compare sorted arrays
        return Arrays.equals(charArray1, charArray2);
    }

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

Answer 8:

Example (Understanding)

```
        // User input
        System.out.print("Enter the first string: ");
        String str1 = scanner.nextLine();

        System.out.print("Enter the second string: ");
        String str2 = scanner.nextLine();

        // Check if they are anagrams
        if (areAnagrams(str1, str2)) {
            System.out.println(str1 + " and " + str2 + " are a
        } else {
            System.out.println(str1 + " and " + str2 + " are N
        }
        scanner.close();
    } }
```

Question 9:

- Write a program to validate an email address using regular expressions.

Answer 9:

Example (Understanding)

```
import java.util.Scanner;
import java.util.regex.Pattern;
import java.util.regex.Matcher;
public class Main {
    private static final String EMAIL_REGEX =
    "^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,6}$";

    public static boolean isValidEmail(String email) {
        // Compile the pattern and match the input email
        Pattern pattern = Pattern.compile(EMAIL_REGEX);
        Matcher matcher = pattern.matcher(email);
        return matcher.matches();
    }
```

Answer 9:

Example (Understanding)

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

        System.out.print("Enter an email address: ");
        String email = scanner.nextLine();

        if (isValidEmail(email)) {
            System.out.println(email +
            " is a valid email address.");
        } else {
            System.out.println(email +
            " is NOT a valid email address.");
        }
        scanner.close();
    }}
```

Question 10:

- Implement a Caesar cipher to encrypt and decrypt a string by shifting characters by a fixed number.

Answer 10:

Example (Understanding)

```
import java.util.Scanner;
public class Main {
    // Encrypt the text by shifting characters forward
    public static String encrypt(String text, int shift) {
        StringBuilder result = new StringBuilder();
        for (char ch : text.toCharArray()) {
            if (Character.isLetter(ch)) {
                char base = Character.isUpperCase(ch) ? ’A’ :
                ch = (char) ((ch - base + shift) % 26 + base);
            }
            result.append(ch);
        }
        return result.toString();
    }
```

Answer 10:

Example (Understanding)

```
// Decrypt the text by shifting characters backward
public static String decrypt(String text, int shift) {
    return encrypt(text, 26 - shift);
}
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    // Input the message
    System.out.print("Enter the message: ");
    String message = scanner.nextLine();
    // Input the shift value
    System.out.print("Enter shift value: ");
```

Answer 10:

Example (Understanding)

```
        int shift = scanner.nextInt();
        String encryptedMessage = encrypt(message, shift);
        System.out.println("Encrypted Message: "
        + encryptedMessage);
        String decryptedMessage = decrypt(encryptedMessage,
        shift);
        System.out.println("Decrypted Message: "
        + decryptedMessage);
        scanner.close();
    }
}
```

Question 11:

- Write a program to validate a password based on the following criteria: At least 8 characters long. Contains at least one uppercase letter, one lowercase letter, one digit, and one special character.

Answer 11:

Example (Understanding)

```
import java.util.Scanner;
public class Main {
    public static boolean isValidPassword(String password) {
        if (password.length() < 8) {
            return false;
        }
        boolean hasUpper = false, hasLower = false,
        hasDigit = false, hasSpecial = false;
        String specialCharacters = "!@#$%^&*()-+";
        for (char ch : password.toCharArray()) {
            if (Character.isUpperCase(ch)) {
                hasUpper = true;
            } else if (Character.isLowerCase(ch)) {
                hasLower = true;
            } else if (Character.isDigit(ch)) {
                hasDigit = true;
```

Answer 11:

Example (Understanding)

```
            } else if (specialCharacters.
            contains(String.valueOf(ch))) {
                hasSpecial = true;
            }
        }
        return hasUpper && hasLower && hasDigit && hasSpecial;
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a password: ");
        String password = scanner.nextLine();
        if (isValidPassword(password)) {
            System.out.println("Password is valid!");
        } else {
```

Answer 11:

Example (Understanding)

```
            System.out.println("Invalid password!
            Make sure it contains at least:");
            System.out.println("- 8 characters");
            System.out.println("- One uppercase letter");
            System.out.println("- One lowercase letter");
            System.out.println("- One digit");
            System.out.println("- One special character
            (!@#$%^&*()-+)");
        }
        scanner.close();
    }
}
```

Question 12:

- Write a program to read a text file, replace all occurrences of a specific word, and write the modified content back to the file.

Answer 12:

Example (Understanding)

```
import java.io.*;
import java.nio.file.*;
public class Main {
    public static void main(String[] args) {
        String filePath = "C:\\Users\\S.A.N\\Desktop\\file.txt
        String oldWord = "Java";
        String newWord = "Python";
        try {
            System.out.println("Attempting to
            access file at: " + filePath);
            Path path = Paths.get(filePath);
            if (!Files.exists(path)) {
                System.out.println("File not found!
                Creating a new file...");
```

Answer 12:

Example (Understanding)

```
    Files.createFile(path);
    System.out.println("File created.
    Please add content and try again.");
    return;
}
// Read file content
String content = new String(Files.
readAllBytes(path));
System.out.println("Original
Content:\n" + content);
// Replace occurrences of the old word
content = content.replaceAll
(oldWord, newWord);
System.out.println("Modified Content:
\n" + content);
```

Answer 12:

Example (Understanding)

```
            // Write modified content back to the file
            Files.write(path, content.getBytes());
            System.out.println("Replacement
            completed successfully!");
        } catch (IOException e) {
            e.printStackTrace(); // Print full error message
        }    }}
```

Question 13:

- Create a custom method to reverse a string without using StringBuilder or any built-in reverse functions.

Answer 13:

Example (Understanding)

```
public class Main {
    public static String reverse(String str) {
        String reversed = "";
        for (int i = str.length() - 1; i >= 0;
        i--) {
            reversed += str.charAt(i);
        }
        return reversed;
    }
    public static void main(String[] args) {
        String input = "Java Programming";
        System.out.println("Original: "
        + input);
        System.out.println("Reversed: "
        + reverse(input));
    }}
```

Wrapper Class

Introduction to Wrapper Classes

- Wrapper Classes provide object representation for primitive data types.
- Useful for working with Collections, Serialization, Multi-threading, etc.

What is a Wrapper Class?

- Java provides built-in classes that ”wrap” primitive data types into objects.
- Example: `Integer` for `int`, `Double` for `double`.

Example (Understanding)

```
int num = 10;  // Primitive type
Integer obj = Integer.valueOf(num);  // Wrapping (Boxing)
int newNum = obj.intValue();  // Unwrapping (Unboxing)
```

Types of Wrapper Classes in Java

- Byte (`Byte`)
- Short (`Short`)
- Integer (`Integer`)
- Long (`Long`)
- Float (`Float`)
- Double (`Double`)
- Character (`Character`)
- Boolean (`Boolean`)

AutoBoxing and Unboxing

- **AutoBoxing**: Automatic conversion of primitive types into their corresponding wrapper classes.
- **Unboxing**: Automatic conversion of wrapper class objects back to their corresponding primitive types.

Example (Understanding)

```
public class AutoBoxingUnboxingExample {
    public static void main(String[] args) {
        Integer obj = 10;  // AutoBoxing
        int num = obj;  // Unboxing
        System.out.println("AutoBoxed: " + obj);
        System.out.println("Unboxed: " + num);
    }
}
```

Wrapper Class Methods

- `valueOf()` - Converts a primitive type to a wrapper object.
- `xxxValue()` (e.g., `intValue()`, `doubleValue()`) - Extracts the primitive value from an object.
- `parseXxx()` (e.g., `parseInt()`, `parseDouble()`) - Converts a String to a primitive type.
- `toString()` - Converts a wrapper object to a String.
- `compareTo()` - Compares two wrapper objects.
- `equals()` - Checks if two wrapper objects are equal.
- `isNaN()`, `isInfinite()` - Used for floating point operations.

valueOf() - Convert Primitive to Wrapper Object

Example (Understanding)

```
int num = 100;
Integer obj = Integer.valueOf(num);
System.out.println(obj);
```

xxxValue() - Extract Primitive Value

Example (Understanding)

```
Double obj = Double.valueOf(99.99);
double val = obj.doubleValue();
System.out.println(val);
```

parseXxx() - Convert String to Primitive

Example (Understanding)

```
String str = "123";
int num = Integer.parseInt(str);
System.out.println(num);
```

toString() - Convert Wrapper Object to String

Example (Understanding)

```
Integer num = Integer.valueOf(456);
String str = num.toString();
System.out.println(str);
```

compareTo() - Compare Two Wrapper Objects

Example (Understanding)

```
Integer num1 = 10;
Integer num2 = 20;
int result = num1.compareTo(num2);
if (result < 0)
    System.out.println("num1 is smaller");
```

equals() - Check Equality

Example (Understanding)

```
Integer num1 = Integer.valueOf(50);
Integer num2 = Integer.valueOf(50);
System.out.println(num1.equals(num2));
```

isNaN() and isInfinite() - Floating Point Checks

Example (Understanding)

```
Double num = Double.valueOf(0.0 / 0.0);
Double infinity = Double.valueOf(1.0 / 0.0);
System.out.println(num.isNaN());
System.out.println(infinity.isInfinite());
```

Why do we need Wrapper Classes?

- Collections Framework: Only objects can be stored in collections.
- Utility Methods: Methods like `parseInt()` and `valueOf()`.
- Default Values: Objects can be assigned `null`, unlike primitives.
- Serialization: Convert objects to byte streams for storage.

What is a Wrapper Class?

Example (Understanding)

```
import java.util.ArrayList;
public class WrapperExample {
    public static void main(String[] args) {
        ArrayList<Integer> numbers = new ArrayList<>();
        numbers.add(10); // Autoboxing
        numbers.add(20);
        System.out.println("Stored numbers: " + numbers);
    }
}
```

Example: Using Wrapper Classes in Collections

Example (Understanding)

```
import java.util.ArrayList;
public class WrapperExample {
    public static void main(String[] args) {
        ArrayList<Integer> numbers = new ArrayList<>();
        numbers.add(10); // Autoboxing
        numbers.add(20);
        System.out.println("Stored numbers: " + numbers);
    }
}
```

Primitive Data Types vs Wrapper Classes

Feature	Primitive Type	Wrapper Class
Memory Usage	Efficient	More overhead
Stored In	Stack	Heap
Null Value Support	No	Yes
Methods	None	Utility methods available
Collections	Not allowed	Allowed
Performance	Faster	Slightly slower

Conversion Between Data Types

- **AutoBoxing**: Converting primitive to wrapper object.
- **Unboxing**: Converting wrapper object to primitive.
- **String to Wrapper**: Using `Integer.valueOf("100")`.
- **Wrapper to String**: Using `toString()` method.
- **One wrapper type to another**: Example: `Double.valueOf(10)`.

Conversion Between Data Types

Example (Understanding)

```java
public class ConversionExample {
    public static void main(String[] args) {
        Integer intObj = 100;
        int intValue = intObj;
        Integer strToInt = Integer.valueOf("100");
        String intToStr = intObj.toString();
        Double doubleObj = Double.valueOf(10);
        System.out.println("AutoBoxing: " + intObj);
        System.out.println("Unboxing: " + intValue);
        System.out.println("String to Wrapper: " + strToInt);
        System.out.println("Wrapper to String: " + intToStr);
        System.out.println("Wrapper to another Wrapper: "
        + doubleObj);
    }}
```

Memory Management in Wrapper Classes

- **Immutable Nature of Wrapper Classes**: Once created, wrapper objects cannot be changed.
- **Caching in Wrapper Classes**: Certain wrapper classes (e.g., `Integer`) cache frequently used values to improve performance.
- **Performance Considerations**: Wrapper objects use more memory than primitives and add overhead.

Example (Understanding)

```
public class IntegerCacheExample {
    public static void main(String[] args) {
        Integer a = 127;
        Integer b = 127;
        System.out.println(a == b);
        Integer x = 128;
        Integer y = 128;
        System.out.println(x == y);
    }}
```

Conclusion

- Wrapper classes enable object representation of primitives.
- Useful in Collections, Multi-threading, and Serialization.
- Provide built-in utility methods for conversions.
- Trade-off: Wrapper classes consume more memory than primitives.

Questions to Practice

- Write a Java program to convert an int primitive to an Integer object using valueOf().
- Create a Java program where a Double wrapper object holds a decimal number. Extract and print the primitive double value using doubleValue().
- Write a program that takes a numeric string (e.g., "345") as input and converts it to an int using parseInt().
- Convert an Integer object to a String using toString(), then print the length of the string.

Questions to Practice

- Write a program that compares two Integer objects using compareTo() and prints whether they are equal, greater, or smaller.
- Take two Double objects with the same value and use equals() to check if they are equal. Print the result.
- Write a program that creates a Double object with NaN and another with Infinity. Use isNaN() and isInfinite() to check their state and print appropriate messages.
- Accept two numbers from the user as String, convert them to Integer objects using valueOf(), perform addition, and print the result.
- Given two numbers stored in Integer objects, use Integer.max() and Integer.min() to find and print the maximum and minimum numbers.
- Write a Java program that demonstrates autoboxing (assigning a primitive to a wrapper) and unboxing (assigning a wrapper to a primitive).

Conclusion

Control Flow Statements: Loops and conditionals manage the execution flow, making programs efficient. Loops (‘for‘, ‘while‘, ‘do-while‘, enhanced ‘for‘) enable iteration, while conditionals (‘if-else‘, ‘switch‘) handle decision-making.

Arrays: Arrays store multiple values efficiently. One-dimensional arrays handle simple lists, while multi-dimensional arrays manage complex data structures like matrices. The enhanced ‘for‘ loop simplifies array traversal.

Strings: Strings in Java are immutable sequences of characters, used for text manipulation. ‘StringBuilder‘ and ‘StringBuffer‘ provide mutable alternatives for performance optimization.

Wrapper Classes: These bridge the gap between primitive types and objects, enabling primitives to work with collections and utility methods. They also offer conversion, comparison, and mathematical functions.

Final Verdict

Mastering these concepts allows:

- Efficient problem-solving.
- Optimized data handling.
- Clean and maintainable Java programs.

Conclusion: These fundamental Java constructs are essential for writing scalable and robust applications. Understanding their applications enhances both coding efficiency and program performance.