Java Useful Classes

Java provides a rich set of built-in classes that simplify common programming tasks. These classes offer powerful features and functionalities for various operations.

AP Spring 1404-Dr. Mojtaba Vahidi Asl
By Zahra Roshani

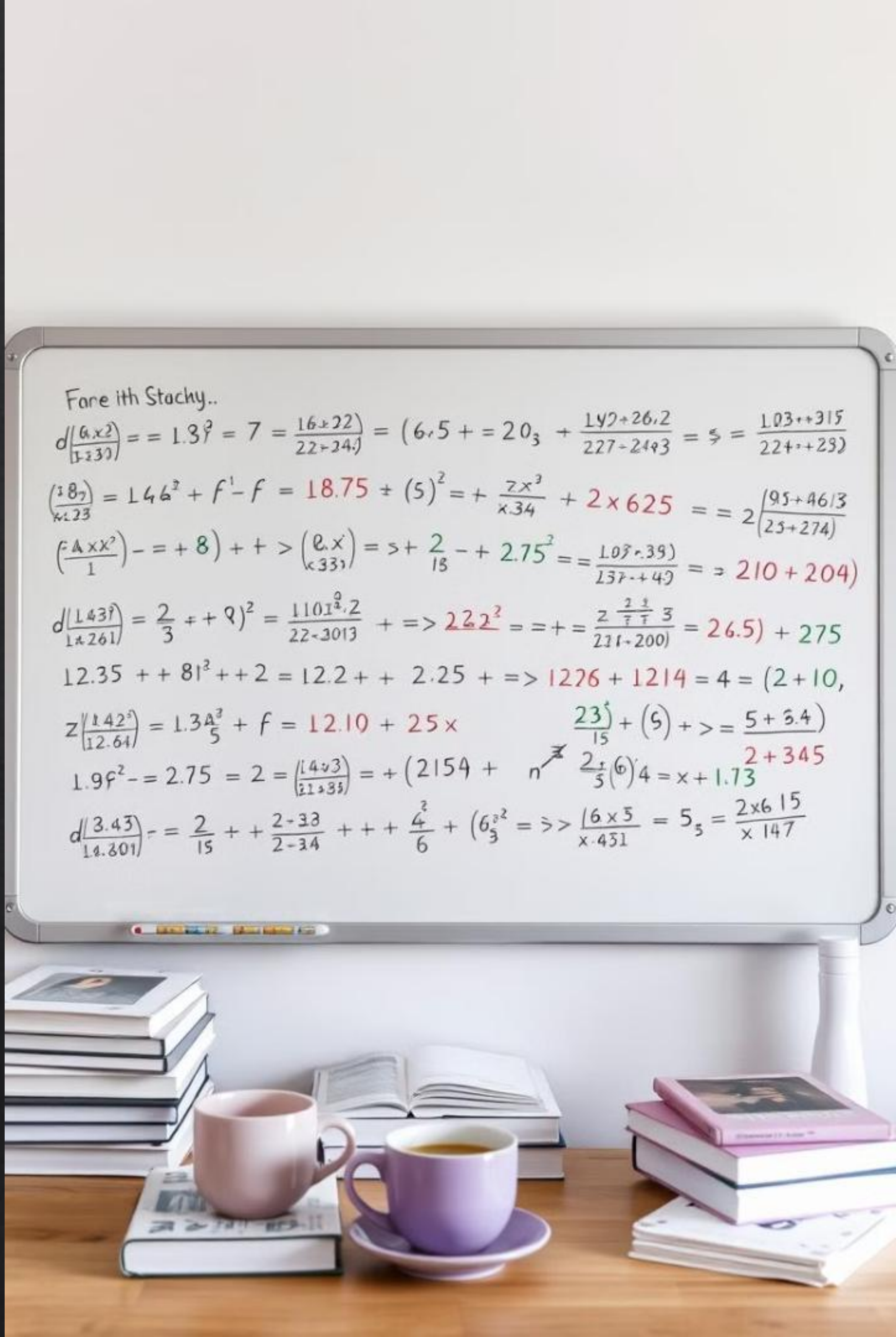
Math Class

The Math class in Java provides numerous mathematical functions.

GeeksforGeeks

Java Math Class - GeeksforGeeks

Learn Math Class in Java to solve the complex mathematical operation, using predefined...



Generating Random Numbers in Java

Math.random() in Java

In Java, the **`Math.random()`** method is a powerful tool for generating random numbers. This method returns a random floating-point number between 0.0 and 1.0.

Generating Random Integers

To generate a random integer within a specific range in Java, you can use the following formula:

```
int randomInt = (int)(Math.random() * (max - min + 1)) + min;
```

This will give you a random integer between **min** and **max** (inclusive).

String Class :

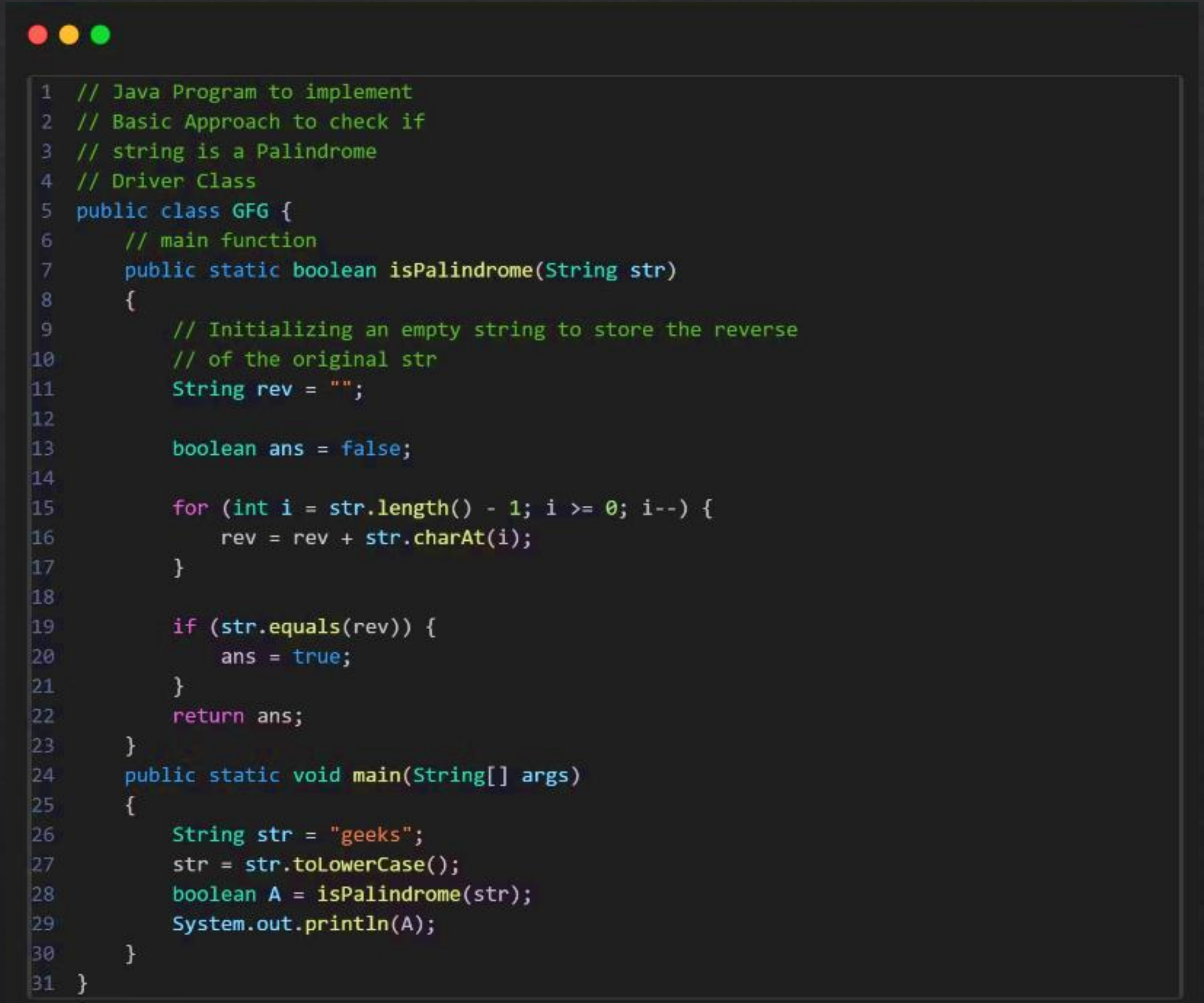
It provides numerous methods for manipulating strings, including searching, comparing, and modifying string content.

www.w3schools.com

W3Schools.com

W3Schools offers free online tutorials, references and exercises in all the...

A string is said to be a palindrome if it is the same if we start reading it from left to right or right to left.


```java
// Java Program to implement
// Basic Approach to check if
// string is a Palindrome
// Driver Class
public class GFG {
    // main function
    public static boolean isPalindrome(String str)
    {
        // Initializing an empty string to store the reverse
        // of the original str
        String rev = "";

        boolean ans = false;

        for (int i = str.length() - 1; i >= 0; i--) {
            rev = rev + str.charAt(i);
        }

        if (str.equals(rev)) {
            ans = true;
        }
        return ans;
    }
    public static void main(String[] args)
    {
        String str = "geeks";
        str = str.toLowerCase();
        boolean A = isPalindrome(str);
        System.out.println(A);
    }
}
```

Mutable vs Immutable Classes

In Java, classes can be classified as mutable or immutable. Mutable classes allow their instances to be modified after creation, while immutable classes prevent any changes to their instances once they're created.

Mutable

- **Mutable objects** are objects whose fields (or state) can be changed after the object has been instantiated.
- Examples of mutable classes include `ArrayList`, `HashMap`, and most other collection classes in Java.

Immutable

- **Immutable objects** are objects whose state cannot be modified after the object is created.
- Examples of immutable classes are `String`, `Integer`, `LocalDate`

Immutability of String Class

The String class in Java is designed to be immutable. This means that once a String object is created, its contents cannot be changed. Any operation that appears to modify a string actually creates a new string object with the updated content, leaving the original string untouched.

1

Memory Efficiency (String Pool):

- Java maintains a **String Pool** for memory optimization. When a new `String` literal is created, the JVM checks if the same value already exists in the pool. If it does, the reference is reused. Immutability ensures that the `String` values in the pool can't be altered by one reference and affect others.

2

Security:

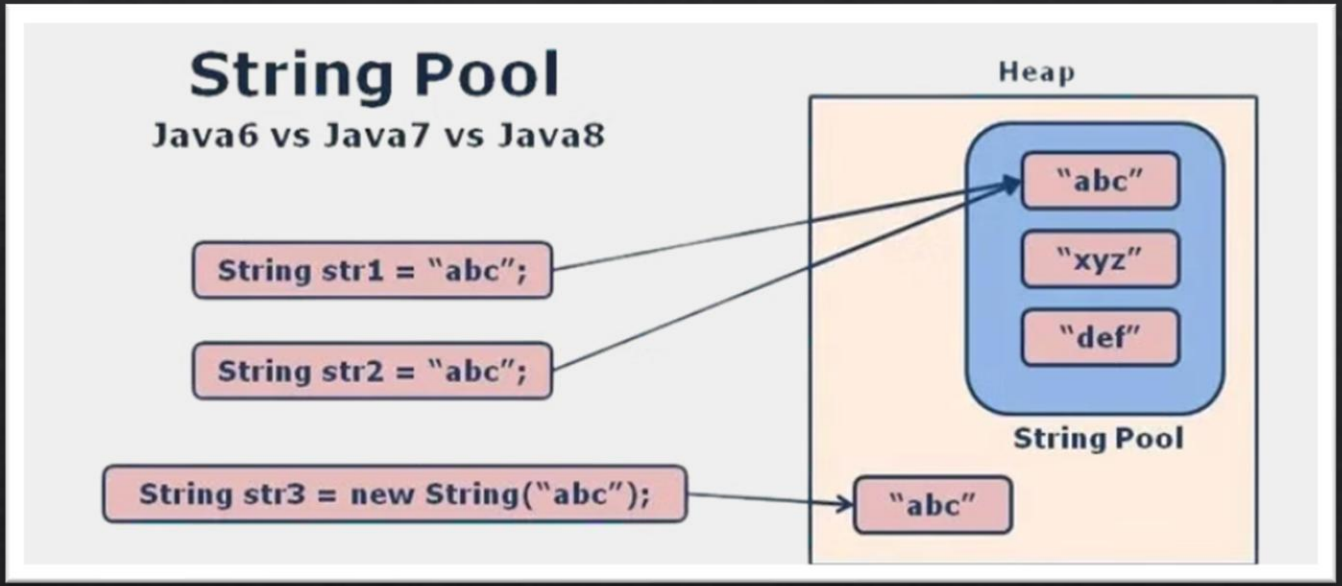
- Strings are widely used to handle sensitive data like passwords, URLs, and file paths. Immutability ensures that once a `String` is created, it cannot be altered by any external code, providing additional security.

3

Thread Safety:

- Since `String` objects cannot be modified, they are inherently **thread-safe**. Multiple threads can safely share and read the same `String` instance without worrying about data corruption or inconsistency.

Understanding Why Strings Are Immutable in Java

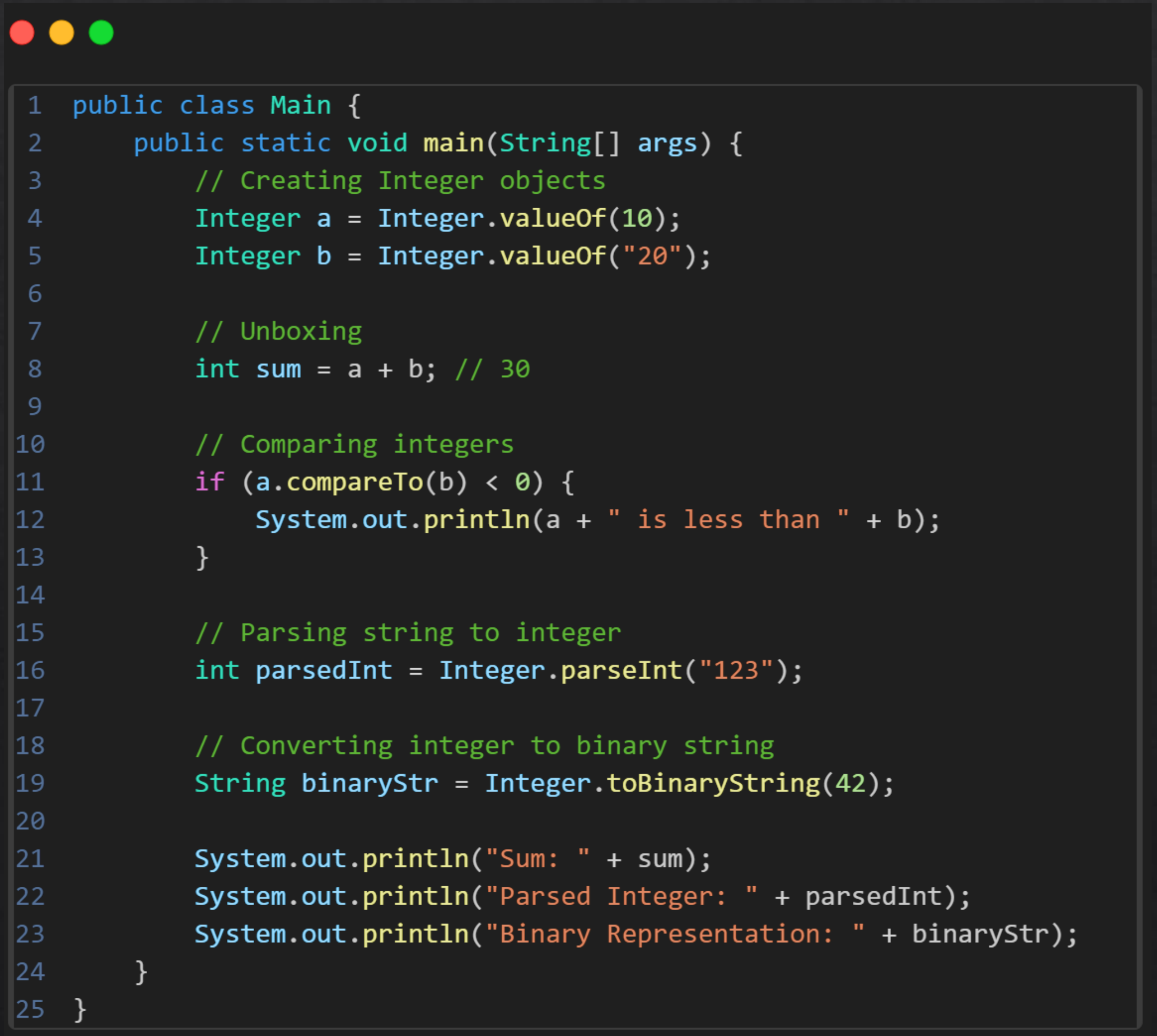


In situations where strings are used to store **sensitive information** like passwords, their immutability ensures that once the password is set, it cannot be changed accidentally or maliciously within the program.

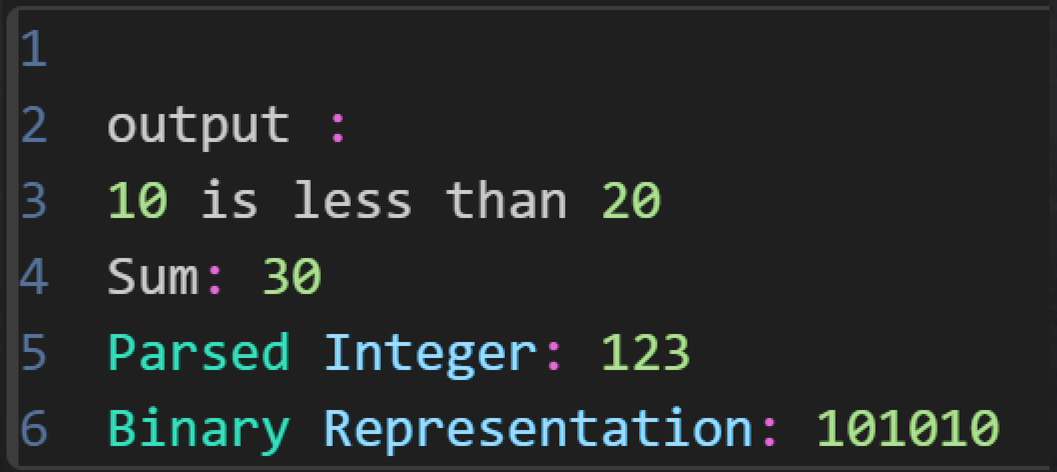
Imagine that you have a thousand customers you wish to email given their first name and email address, 30% of these customers have the same first name.

Integer Class:

The Integer class in Java provides a wrapper for the primitive data type int. The Integer class is essential for handling **integer values as objects**, enabling operations that are not directly available for primitive data types.



```java
public class Main {
    public static void main(String[] args) {
        // Creating Integer objects
        Integer a = Integer.valueOf(10);
        Integer b = Integer.valueOf("20");

        // Unboxing
        int sum = a + b; // 30

        // Comparing integers
        if (a.compareTo(b) < 0) {
            System.out.println(a + " is less than " + b);
        }

        // Parsing string to integer
        int parsedInt = Integer.parseInt("123");

        // Converting integer to binary string
        String binaryStr = Integer.toBinaryString(42);

        System.out.println("Sum: " + sum);
        System.out.println("Parsed Integer: " + parsedInt);
        System.out.println("Binary Representation: " + binaryStr);
    }
}
```



```
output :
10 is less than 20
Sum: 30
Parsed Integer: 123
Binary Representation: 101010
```

Character Class

The Character class in Java provides methods for working with individual characters. These methods include testing character properties, converting characters, and performing basic character operations. The Character class simplifies working with character data, allowing for efficient manipulation and analysis of individual characters.

Character Properties

Determine if a character is uppercase, lowercase, a digit, or whitespace.

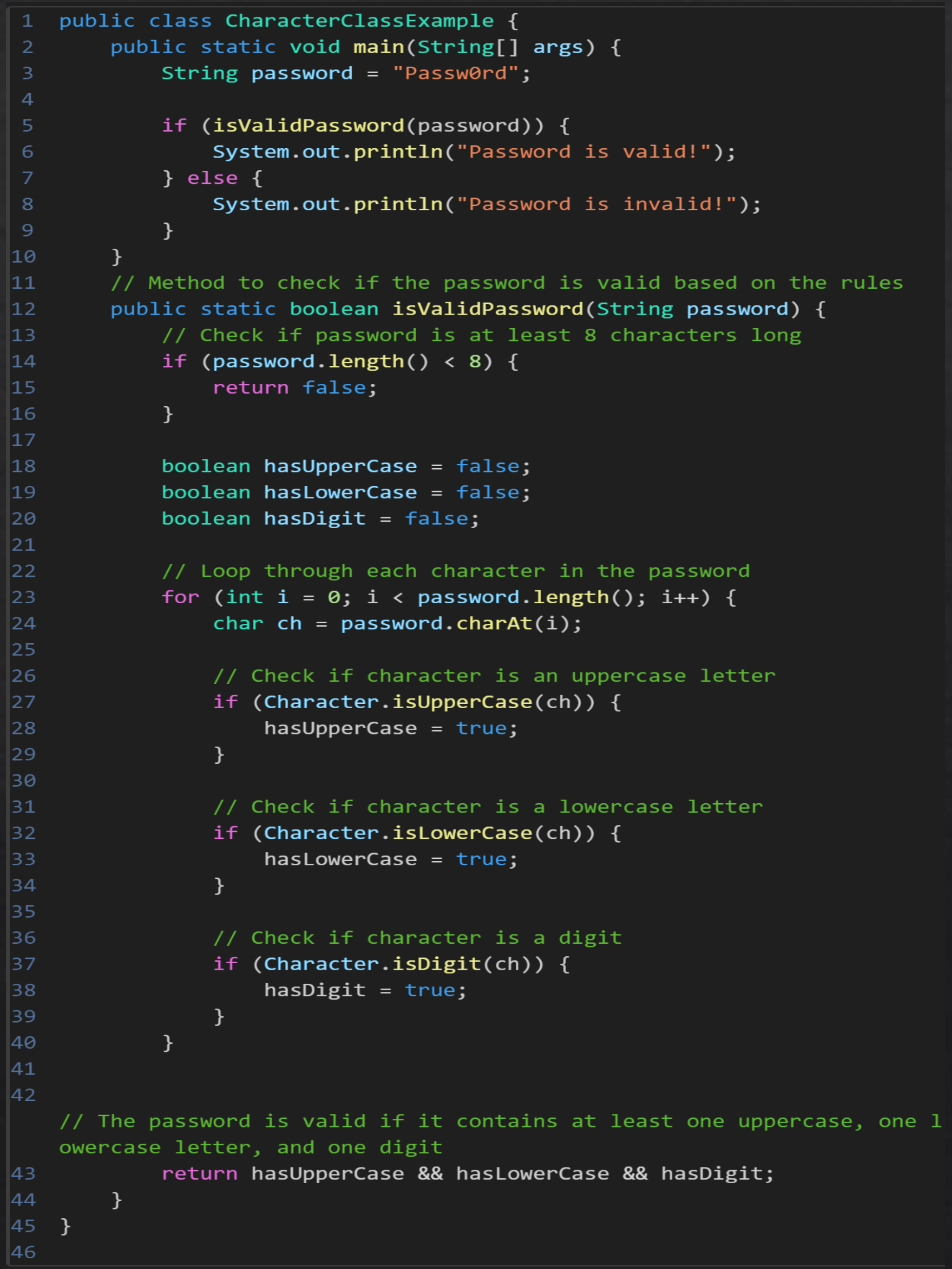
Character Conversion

Convert characters between uppercase and lowercase, or from characters to integers and vice versa.

Character Comparison

Compare characters lexicographically and determine their relative order.

Functional Example: validating a password



```java
public class CharacterClassExample {
    public static void main(String[] args) {
        String password = "Passw0rd";

        if (isValidPassword(password)) {
            System.out.println("Password is valid!");
        } else {
            System.out.println("Password is invalid!");
        }
    }
    // Method to check if the password is valid based on the rules
    public static boolean isValidPassword(String password) {
        // Check if password is at least 8 characters long
        if (password.length() < 8) {
            return false;
        }

        boolean hasUpperCase = false;
        boolean hasLowerCase = false;
        boolean hasDigit = false;

        // Loop through each character in the password
        for (int i = 0; i < password.length(); i++) {
            char ch = password.charAt(i);

            // Check if character is an uppercase letter
            if (Character.isUpperCase(ch)) {
                hasUpperCase = true;
            }

            // Check if character is a lowercase letter
            if (Character.isLowerCase(ch)) {
                hasLowerCase = true;
            }

            // Check if character is a digit
            if (Character.isDigit(ch)) {
                hasDigit = true;
            }
        }

// The password is valid if it contains at least one uppercase, one lowercase letter, and one digit
        return hasUpperCase && hasLowerCase && hasDigit;
    }
}
```

StringBuilder

The StringBuilder class in Java provides a **mutable** sequence of characters. Unlike String, which is immutable, StringBuilder allows for efficient modification of string content through various methods. This makes StringBuilder ideal for building strings dynamically or when frequent modifications are required.

1 Insert

Insert a string at a specific position within the StringBuilder.

2 Append

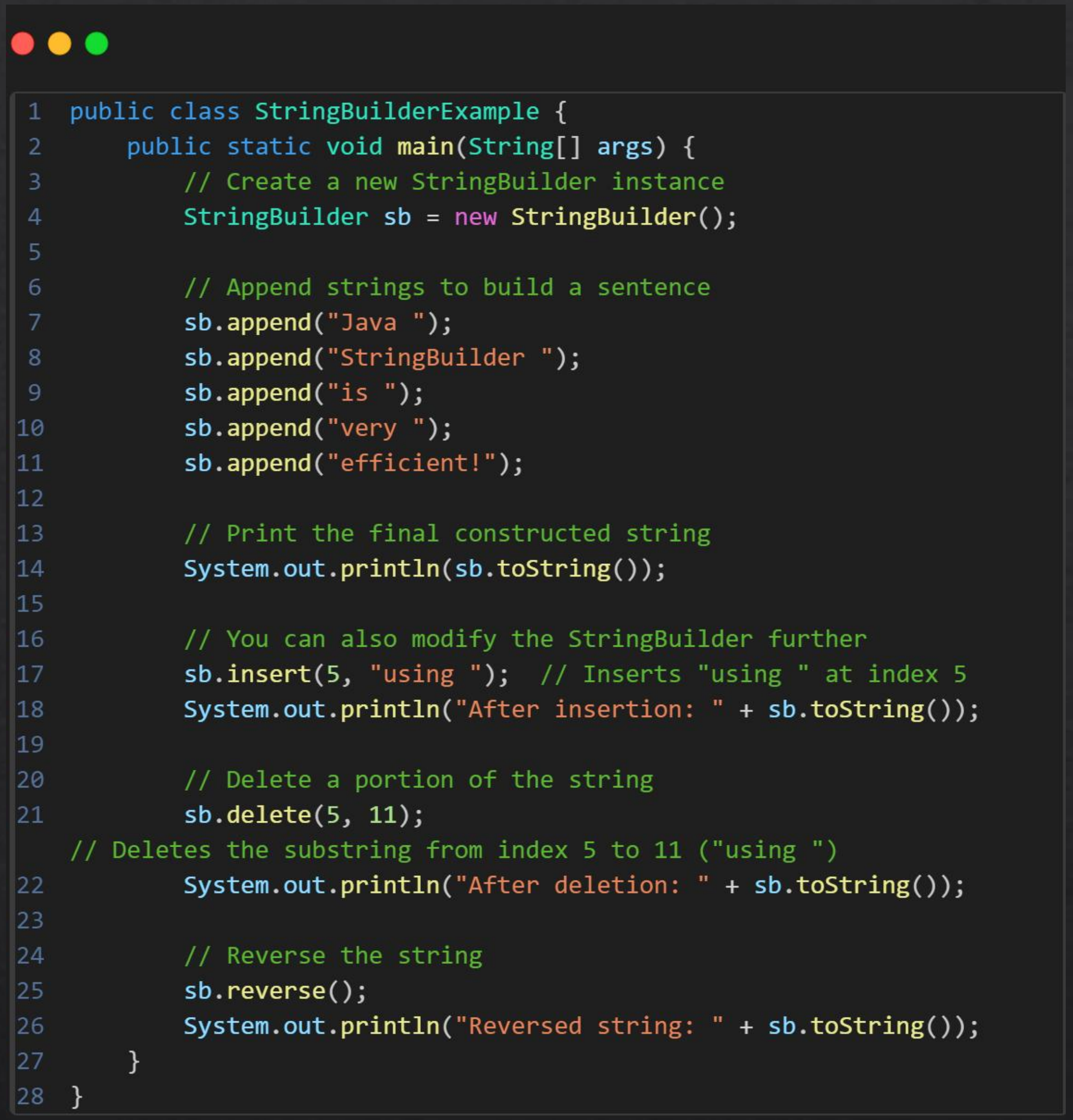
Add a string to the end of the StringBuilder.

3 Reverse

Reverse the order of characters within the StringBuilder.

4 Replace

Replace all occurrences of a specific character with another character.



```java
public class StringBuilderExample {
    public static void main(String[] args) {
        // Create a new StringBuilder instance
        StringBuilder sb = new StringBuilder();

        // Append strings to build a sentence
        sb.append("Java ");
        sb.append("StringBuilder ");
        sb.append("is ");
        sb.append("very ");
        sb.append("efficient!");

        // Print the final constructed string
        System.out.println(sb.toString());

        // You can also modify the StringBuilder further
        sb.insert(5, "using ");  // Inserts "using " at index 5
        System.out.println("After insertion: " + sb.toString());

        // Delete a portion of the string
        sb.delete(5, 11);
// Deletes the substring from index 5 to 11 ("using ")
        System.out.println("After deletion: " + sb.toString());

        // Reverse the string
        sb.reverse();
        System.out.println("Reversed string: " + sb.toString());
    }
}
```

```
output:
Java StringBuilder is very efficient!
After insertion: Java using StringBuilder is very efficient!
After deletion: Java StringBuilder is very efficient!
Reversed string: !tneiciffe yrev si redliuBgnirtS avaJ
```

Resources

Java Math Class – GeeksforGeeks

Java String Reference (w3schools.com)

https://codechunkers.medium.com/understanding-theimmutability-of-strings-in-java-9c1b973c303

https://youtu.be/Bj9Mx_Lx3q4?si=3NmYbatlLCPOHAFY

https://www.geeksforgeeks.org/stringbuilder-class-in-java-with-examples/

Any Questions?

Thank You for Your Attention