Task: Create a program that demonstrates common String methods for text analysis and
manipulation.

```
public class StringBuiltInMethods {
public static void main(String[] args) {
String sampleText = " Java Programming is Fun and Challenging! ";
// TODO: Use built-in methods to perform the following operations:
// 1. Display original string length including spaces
// 2. Remove leading and trailing spaces, show new length
// 3. Find and display the character at index 5
// 4. Extract substring "Programming" from the text
// 5. Find the index of the word "Fun"
// 6. Check if the string contains "Java" (case-sensitive)
// 7. Check if the string starts with "Java" (after trimming)
// 8. Check if the string ends with an exclamation mark
// 9. Convert the entire string to uppercase
// 10. Convert the entire string to lowercase
// TODO: Create a method that counts vowels using charAt()
// TODO: Create a method that finds all occurrences of a character
// TODO: Display all results in a formatted manner
}
// TODO: Method to count vowels in a string
public static int countVowels(String text) {
// Your code here
}
// TODO: Method to find all positions of a character
public static void findAllOccurrences(String text, char target) {
// Your code here
}
}
```

```
public class StringBuiltInMethods {
    public static void main(String[] args) {
        String sampleText = " Java Programming is Fun and Challenging! ";

        // 1. Display original string length including spaces
        System.out.println("1. Original String: \"" + sampleText + "\"");
        System.out.println("   Original Length (with spaces): " + sampleText.length());

        // 2. Remove leading and trailing spaces, show new length
        String trimmedText = sampleText.trim();
        System.out.println("\n2. Trimmed String: \"" + trimmedText + "\"");
        System.out.println("   New Length (without spaces): " + trimmedText.length());

        // 3. Find and display the character at index 5
        System.out.println("\n3. Character at index 5: " + sampleText.charAt(5));

        // 4. Extract substring "Programming" from the text
```

```
    String substring = trimmedText.substring(5, 16); // "Programming"
    System.out.println("\n4. Substring (\"Programming\"): " + substring);

    // 5. Find the index of the word "Fun"
    int funIndex = trimmedText.indexOf("Fun");
    System.out.println("\n5. Index of \"Fun\": " + funIndex);

    // 6. Check if the string contains "Java" (case-sensitive)
    System.out.println("\n6. Contains \"Java\"? " + trimmedText.contains("Java"));

    // 7. Check if the string starts with "Java" (after trimming)
    System.out.println("\n7. Starts with \"Java\"? " + trimmedText.startsWith("Java"));

    // 8. Check if the string ends with an exclamation mark
    System.out.println("\n8. Ends with \"!\"? " + trimmedText.endsWith("!"));

    // 9. Convert the entire string to uppercase
    System.out.println("\n9. Uppercase: " + trimmedText.toUpperCase());

    // 10. Convert the entire string to lowercase
    System.out.println("\n10. Lowercase: " + trimmedText.toLowerCase());

    // Count vowels
    int vowelCount = countVowels(trimmedText);
    System.out.println("\n11. Total Vowels: " + vowelCount);

    // Find all occurrences of 'a'
    System.out.println("\n12. Occurrences of 'a':");
    findAllOccurrences(trimmedText, 'a');
}

// Method to count vowels in a string
public static int countVowels(String text) {
    int count = 0;
    text = text.toLowerCase();
    for (int i = 0; i < text.length(); i++) {
        char ch = text.charAt(i);
        if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {
            count++;
        }
    }
    return count;
}

// Method to find all positions of a character
public static void findAllOccurrences(String text, char target) {
    boolean found = false;
    for (int i = 0; i < text.length(); i++) {
```

```
            if (text.charAt(i) == target) {
                System.out.println("   '" + target + "' found at index: " + i);
                found = true;
            }
        }
        if (!found) {
            System.out.println("   Character '" + target + "' not found in text.");
        }
    }
}
```

String Manipulation Methods
Task: Create a text processing utility that uses various string manipulation methods.

```
import java.util.Scanner;
public class StringManipulation {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
// TODO: Ask user to enter a sentence with mixed formatting
// TODO: Process the input using the following methods:
// 1. trim() - Remove extra spaces
// 2. replace() - Replace all spaces with underscores
// 3. replaceAll() - Remove all digits using regex
// 4. split() - Split sentence into words array
// 5. join() - Rejoin words with " | " separator
// TODO: Create additional processing methods:
// - Remove all punctuation
// - Capitalize first letter of each word
// - Reverse the order of words
// - Count word frequency
scanner.close();
}
// TODO: Method to remove punctuation
public static String removePunctuation(String text) {
// Your code here
}
// TODO: Method to capitalize each word
public static String capitalizeWords(String text) {
// Your code here
}
// TODO: Method to reverse word order
```

2

```
public static String reverseWordOrder(String text) {
// Your code here
}
// TODO: Method to count word frequency
public static void countWordFrequency(String text) {
```

```
// Your code here
}
}
import java.util.Scanner;
import java.util.Arrays;
import java.util.HashMap;

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

        // Ask user to enter a sentence with mixed formatting
        System.out.println("Enter a sentence with mixed formatting:");
        String input = scanner.nextLine();

        // 1. trim() - Remove extra spaces at beginning and end
        String trimmed = input.trim();
        System.out.println("\n1. Trimmed String: " + trimmed);

        // 2. replace() - Replace all spaces with underscores
        String replacedSpaces = trimmed.replace(" ", "_");
        System.out.println("\n2. Replace spaces with underscores: " + replacedSpaces);

        // 3. replaceAll() - Remove all digits using regex
        String noDigits = trimmed.replaceAll("\\d", "");
        System.out.println("\n3. Removed digits: " + noDigits);

        // 4. split() - Split sentence into words array
        String[] words = trimmed.split("\\s+");
        System.out.println("\n4. Words Array: " + Arrays.toString(words));

        // 5. join() - Rejoin words with " | " separator
        String rejoined = String.join(" | ", words);
        System.out.println("\n5. Rejoined with | : " + rejoined);

        // Additional Processing
        System.out.println("\n--- Additional Processing ---");

        // Remove punctuation
        String noPunct = removePunctuation(trimmed);
        System.out.println("Removed punctuation: " + noPunct);

        // Capitalize each word
        String capitalized = capitalizeWords(noPunct);
        System.out.println("Capitalized words: " + capitalized);

        // Reverse word order
        String reversedOrder = reverseWordOrder(noPunct);
```

```
    System.out.println("Reversed word order: " + reversedOrder);

    // Count word frequency
    System.out.println("\nWord Frequencies:");
    countWordFrequency(noPunct);

    scanner.close();
}

// Method to remove punctuation
public static String removePunctuation(String text) {
    return text.replaceAll("[^a-zA-Z0-9\\s]", ""); // keep only letters, numbers, and spaces
}

// Method to capitalize each word
public static String capitalizeWords(String text) {
    String[] words = text.split("\\s+");
    StringBuilder sb = new StringBuilder();
    for (String word : words) {
        if (!word.isEmpty()) {
            sb.append(Character.toUpperCase(word.charAt(0)))
              .append(word.substring(1).toLowerCase())
              .append(" ");
        }
    }
    return sb.toString().trim();
}

// Method to reverse word order
public static String reverseWordOrder(String text) {
    String[] words = text.split("\\s+");
    StringBuilder sb = new StringBuilder();
    for (int i = words.length - 1; i >= 0; i--) {
        sb.append(words[i]).append(" ");
    }
    return sb.toString().trim();
}

// Method to count word frequency
public static void countWordFrequency(String text) {
    String[] words = text.toLowerCase().split("\\s+");
    HashMap<String, Integer> freqMap = new HashMap<>();
    for (String word : words) {
        freqMap.put(word, freqMap.getOrDefault(word, 0) + 1);
    }
    for (String word : freqMap.keySet()) {
        System.out.println("    " + word + " : " + freqMap.get(word));
    }
```

```
    }
}
```

ASCII Codes and Character Conversion
Task: Create a program that demonstrates ASCII character manipulation and conversion.

```
import java.util.Scanner;
public class ASCIIProcessor {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
// TODO: Ask user to enter a string
// TODO: For each character in the string:
// 1. Display the character and its ASCII code
// 2. Determine if it's uppercase, lowercase, digit, or special
character
// 3. If letter, show both upper and lower case versions with ASCII
codes
// 4. Calculate the difference between upper and lower case ASCII
values
// TODO: Create ASCII art using character codes
// TODO: Implement a simple Caesar cipher using ASCII manipulation
scanner.close();
}
```

3

```
// TODO: Method to classify character type
public static String classifyCharacter(char ch) {
// Return "Uppercase Letter", "Lowercase Letter", "Digit", or
"Special Character"
// Your code here
}
// TODO: Method to convert case using ASCII manipulation
public static char toggleCase(char ch) {
// Convert upper to lower and lower to upper using ASCII values
// Your code here
}
// TODO: Method to implement Caesar cipher
public static String caesarCipher(String text, int shift) {
// Shift each letter by 'shift' positions in ASCII
// Your code here
}
// TODO: Method to create ASCII table for a range
public static void displayASCIITable(int start, int end) {
// Display ASCII codes and corresponding characters
// Your code here
}
```

```
// TODO: Method to convert string to ASCII array
public static int[] stringToASCII(String text) {
// Your code here
}
// TODO: Method to convert ASCII array back to string
public static String asciiToString(int[] asciiValues) {
// Your code here
}
}
import java.util.Scanner;

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

        // Ask user to enter a string
        System.out.print("Enter a string: ");
        String input = scanner.nextLine();

        System.out.println("\n--- Character Analysis ---");
        for (int i = 0; i < input.length(); i++) {
            char ch = input.charAt(i);
            int ascii = (int) ch;

            System.out.println("Character: '" + ch + "' | ASCII: " + ascii);
            System.out.println("Type: " + classifyCharacter(ch));

            // If it's a letter, show both versions and ASCII diff
            if (Character.isLetter(ch)) {
                char toggled = toggleCase(ch);
                System.out.println("Toggle Case: '" + toggled + "' | ASCII: " + (int) toggled);

                int diff = Math.abs(((int) Character.toUpperCase(ch)) - ((int)
Character.toLowerCase(ch)));
                System.out.println("ASCII difference between upper & lower case: " + diff);
            }
            System.out.println("------------------------");
        }

        // Show ASCII table for A–Z
        System.out.println("\n--- ASCII Table (A–Z) ---");
        displayASCIITable(65, 90);

        // Convert string to ASCII array and back
        System.out.println("\n--- String to ASCII Array ---");
        int[] asciiArray = stringToASCII(input);
        for (int code : asciiArray) {
            System.out.print(code + " ");
```

```
    }
    System.out.println("\nConverted back: " + asciiToString(asciiArray));

    // Caesar cipher demo
    System.out.println("\n--- Caesar Cipher ---");
    System.out.print("Enter shift value: ");
    int shift = scanner.nextInt();
    String encrypted = caesarCipher(input, shift);
    System.out.println("Encrypted: " + encrypted);
    System.out.println("Decrypted: " + caesarCipher(encrypted, -shift));

    scanner.close();
}

// Method to classify character type
public static String classifyCharacter(char ch) {
    if (ch >= 'A' && ch <= 'Z') return "Uppercase Letter";
    else if (ch >= 'a' && ch <= 'z') return "Lowercase Letter";
    else if (ch >= '0' && ch <= '9') return "Digit";
    else return "Special Character";
}

// Method to convert case using ASCII manipulation
public static char toggleCase(char ch) {
    if (ch >= 'A' && ch <= 'Z') {
        return (char) (ch + 32); // Upper -> Lower
    } else if (ch >= 'a' && ch <= 'z') {
        return (char) (ch - 32); // Lower -> Upper
    }
    return ch; // Non-alphabet remains same
}

// Method to implement Caesar cipher
public static String caesarCipher(String text, int shift) {
    StringBuilder result = new StringBuilder();
    for (char ch : text.toCharArray()) {
        if (Character.isLetter(ch)) {
            char base = Character.isUpperCase(ch) ? 'A' : 'a';
            char shifted = (char) ((ch - base + shift + 26) % 26 + base);
            result.append(shifted);
        } else {
            result.append(ch);
        }
    }
    return result.toString();
}

// Method to display ASCII table
```

```
    public static void displayASCIITable(int start, int end) {
        for (int i = start; i <= end; i++) {
            System.out.println(i + " : " + (char) i);
        }
    }

    // Method to convert string to ASCII array
    public static int[] stringToASCII(String text) {
        int[] ascii = new int[text.length()];
        for (int i = 0; i < text.length(); i++) {
            ascii[i] = (int) text.charAt(i);
        }
        return ascii;
    }

    // Method to convert ASCII array back to string
    public static String asciiToString(int[] asciiValues) {
        StringBuilder sb = new StringBuilder();
        for (int val : asciiValues) {
            sb.append((char) val);
        }
        return sb.toString();
    }
}
```

StringBuilder, StringBuffer, and Performance
Task: Create a performance comparison program that demonstrates the differences between
String, StringBuilder, and StringBuffer.

```
public class StringPerformanceComparison {
public static void main(String[] args) {
// TODO: Implement performance tests for different approaches
// Test 1: String concatenation performance
System.out.println("=== PERFORMANCE COMPARISON ===");
// TODO: Test string concatenation with regular String (slow method)
long startTime = System.nanoTime();
String result1 = concatenateWithString(1000);
long endTime = System.nanoTime();
System.out.println("String concatenation time: " + (endTime - startTime) + " ns");
// TODO: Test string concatenation with StringBuilder (fast method)
// TODO: Test string concatenation with StringBuffer (thread-safe method)
// TODO: Compare memory usage (approximate)
// TODO: Demonstrate thread safety differences
// TODO: Create practical examples showing when to use each approach
```

```
}
// TODO: Method using String concatenation (inefficient)
public static String concatenateWithString(int iterations) {
String result = "";
for (int i = 0; i < iterations; i++) {
result += "Java " + i + " ";
}

5

return result;
}
// TODO: Method using StringBuilder (efficient, not thread-safe)
public static String concatenateWithStringBuilder(int iterations) {
// Your code here
}
// TODO: Method using StringBuffer (efficient, thread-safe)
public static String concatenateWithStringBuffer(int iterations) {
// Your code here
}
// TODO: Method to demonstrate StringBuilder methods
public static void demonstrateStringBuilderMethods() {
StringBuilder sb = new StringBuilder("Hello World");
// TODO: Use the following StringBuilder methods:
// 1. append() - Add text to end
// 2. insert() - Insert text at specific position
// 3. delete() - Remove characters from range
// 4. deleteCharAt() - Remove character at index
// 5. reverse() - Reverse the string
// 6. replace() - Replace substring
// 7. setCharAt() - Change character at index
// 8. capacity() - Show current capacity
// 9. ensureCapacity() - Set minimum capacity
// 10. trimToSize() - Reduce capacity to current length
// Your code here
}
// TODO: Method to demonstrate StringBuffer thread safety
public static void demonstrateThreadSafety() {
// Create multiple threads that modify the same StringBuffer
// Show that StringBuffer is thread-safe while StringBuilder is not
// Your code here
}
// TODO: Method to compare string comparison methods

6

public static void compareStringComparisonMethods() {
String str1 = "Hello";
```

```
String str2 = "Hello";
String str3 = new String("Hello");
// TODO: Compare using:
// 1. == operator (reference comparison)
// 2. equals() method (content comparison)
// 3. equalsIgnoreCase() method
// 4. compareTo() method (lexicographic comparison)
// 5. compareToIgnoreCase() method
// TODO: Explain the differences and when to use each
// Your code here
}
// TODO: Method to demonstrate memory efficiency
public static void demonstrateMemoryEfficiency() {
// TODO: Show memory usage before and after different string
operations
// TODO: Demonstrate string pool behavior
// TODO: Show StringBuilder capacity management
// Your code here
}
}
public class StringPerformanceComparison {
    public static void main(String[] args) {
        System.out.println("=== PERFORMANCE COMPARISON ===");

        // Test 1: String concatenation
        long startTime = System.nanoTime();
        String result1 = concatenateWithString(1000);
        long endTime = System.nanoTime();
        System.out.println("String concatenation time: " + (endTime - startTime) + " ns");

        // Test 2: StringBuilder
        startTime = System.nanoTime();
        String result2 = concatenateWithStringBuilder(1000);
        endTime = System.nanoTime();
        System.out.println("StringBuilder concatenation time: " + (endTime - startTime) + " ns");

        // Test 3: StringBuffer
        startTime = System.nanoTime();
        String result3 = concatenateWithStringBuffer(1000);
        endTime = System.nanoTime();
        System.out.println("StringBuffer concatenation time: " + (endTime - startTime) + " ns");

        // Demonstrate StringBuilder methods
        System.out.println("\n=== StringBuilder Methods Demo ===");
        demonstrateStringBuilderMethods();

        // Demonstrate Thread Safety difference
        System.out.println("\n=== Thread Safety Demo ===");
```

```
        demonstrateThreadSafety();

        // String comparison methods
        System.out.println("\n=== String Comparison Demo ===");
        compareStringComparisonMethods();

        // Memory efficiency demo
        System.out.println("\n=== Memory Efficiency Demo ===");
        demonstrateMemoryEfficiency();
    }

    // Method using String concatenation (inefficient)
    public static String concatenateWithString(int iterations) {
        String result = "";
        for (int i = 0; i < iterations; i++) {
            result += "Java " + i + " ";
        }
        return result;
    }

    // Method using StringBuilder (efficient, not thread-safe)
    public static String concatenateWithStringBuilder(int iterations) {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < iterations; i++) {
            sb.append("Java ").append(i).append(" ");
        }
        return sb.toString();
    }

    // Method using StringBuffer (efficient, thread-safe)
    public static String concatenateWithStringBuffer(int iterations) {
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < iterations; i++) {
            sb.append("Java ").append(i).append(" ");
        }
        return sb.toString();
    }

    // Demonstrate StringBuilder methods
    public static void demonstrateStringBuilderMethods() {
        StringBuilder sb = new StringBuilder("Hello World");

        sb.append(" Java");              // append
        sb.insert(6, "Beautiful ");      // insert
        sb.delete(6, 16);                // delete range
        sb.deleteCharAt(5);              // deleteCharAt
        sb.reverse();                    // reverse
        sb.reverse();                    // reverse back
```

```
        sb.replace(0, 5, "Hi");          // replace
        sb.setCharAt(3, 'X');            // setCharAt
        System.out.println("StringBuilder content: " + sb);

        System.out.println("Capacity: " + sb.capacity());
        sb.ensureCapacity(50);
        System.out.println("New Capacity: " + sb.capacity());
        sb.trimToSize();
        System.out.println("Trimmed Capacity: " + sb.capacity());
    }

    // Demonstrate thread safety
    public static void demonstrateThreadSafety() {
        StringBuffer safeBuffer = new StringBuffer("Safe");
        StringBuilder unsafeBuilder = new StringBuilder("Unsafe");

        Runnable bufferTask = () -> {
            for (int i = 0; i < 1000; i++) {
                safeBuffer.append("X");
            }
        };

        Runnable builderTask = () -> {
            for (int i = 0; i < 1000; i++) {
                unsafeBuilder.append("X");
            }
        };

        Thread t1 = new Thread(bufferTask);
        Thread t2 = new Thread(bufferTask);
        Thread t3 = new Thread(builderTask);
        Thread t4 = new Thread(builderTask);

        t1.start(); t2.start(); t3.start(); t4.start();

        try {
            t1.join(); t2.join(); t3.join(); t4.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("StringBuffer length (thread-safe): " + safeBuffer.length());
        System.out.println("StringBuilder length (NOT thread-safe, may vary): " +
unsafeBuilder.length());
    }

    // Compare string comparison methods
    public static void compareStringComparisonMethods() {
```

```
        String str1 = "Hello";
        String str2 = "Hello";
        String str3 = new String("Hello");

        System.out.println("str1 == str2: " + (str1 == str2)); // true (same pool reference)
        System.out.println("str1 == str3: " + (str1 == str3)); // false (different objects)
        System.out.println("str1.equals(str3): " + str1.equals(str3)); // true (content check)
        System.out.println("str1.equalsIgnoreCase(\"hello\"): " +
str1.equalsIgnoreCase("hello")); // true
        System.out.println("str1.compareTo(str3): " + str1.compareTo(str3)); // 0 (equal)
        System.out.println("str1.compareToIgnoreCase(\"hello\"): " +
str1.compareToIgnoreCase("hello")); // 0
    }

    // Demonstrate memory efficiency
    public static void demonstrateMemoryEfficiency() {
        String s1 = "Hello";
        String s2 = "Hello";  // Reuses same pool object
        String s3 = new String("Hello"); // Creates new object

        System.out.println("s1 == s2 (string pool): " + (s1 == s2));
        System.out.println("s1 == s3 (heap object): " + (s1 == s3));

        StringBuilder sb = new StringBuilder("Hello");
        System.out.println("Initial capacity: " + sb.capacity()); // default 16 + length
        sb.append(" This is a longer string to check capacity expansion...");
        System.out.println("Expanded capacity: " + sb.capacity());
    }
}
```