Lab 5

Contents:

String - string concatenation, tostring, string conversion, character extraction, string comparison, searching, substring, replace, trim, strip, stringbuffer; Java Math, Scanner - scanner hasnext and next methods; ArrayList - arraylist, obtaining array from list, methods of arraylist.

Background:

String Methods:

- `charAt(int index)` - Returns the character at the specified index in the string.
- `compareTo(String str)` - Compares this string to the specified string lexicographically.
- `concat(String str)` - Concatenates the specified string to the end of this string.
- `contains(CharSequence s)` - Returns true if this string contains the specified sequence of char values.
- `endsWith(String suffix)` - Tests if this string ends with the specified suffix.
- `equals(Object anObject)` - Compares this string to the specified object.
- `equalsIgnoreCase(String anotherString)` - Compares this string to another string, ignoring case considerations.
- `indexOf(int ch)` - Returns the index within this string of the first occurrence of the specified character.
- `indexOf(int ch, int fromIndex)` - Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
- `indexOf(String str)` - Returns the index within this string of the first occurrence of the specified substring.
- `indexOf(String str, int fromIndex)` - Returns the index within this string of the first occurrence of the specified substring, starting the search at the specified index.
- `isEmpty()` - Returns true if, and only if, length() is 0.
- `lastIndexOf(int ch)` - Returns the index within this string of the last occurrence of the specified character.
- `lastIndexOf(int ch, int fromIndex)` - Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
- `lastIndexOf(String str)` - Returns the index within this string of the last occurrence of the specified substring.
- `lastIndexOf(String str, int fromIndex)` - Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.
- `length()` - Returns the length of this string.
- `replace(char oldChar, char newChar)` - Returns a new string resulting from

replacing all occurrences of oldChar in this string with newChar.

- `replaceAll(String regex, String replacement)` - Replaces each substring of this string that matches the given regular expression with the given replacement.
- `replaceFirst(String regex, String replacement)` - Replaces the first substring of this string that matches the given regular expression with the given replacement.
- `split(String regex)` - Splits this string around matches of the given regular expression.
- `startsWith(String prefix)` - Tests if this string starts with the specified prefix.
- `substring(int beginIndex)` - Returns a new string that is a substring of this string.
- `substring(int beginIndex, int endIndex)` - Returns a new string that is a substring of this string.
- `toCharArray()` - Converts this string to a new character array.
- `toLowerCase()` - Converts all of the characters in this string to lower case.
- `toUpperCase()` - Converts all of the characters in this string to upper case.
- `trim()` - Returns a copy of the string, with leading and trailing whitespace omitted.

String Concatenation:

```
public class StringConcatenationExercise {
	public static void main(String[] args) {
	String firstName = "John";
	String lastName = "Doe";

	String fullName = firstName + " " + lastName;
	System.out.println("Full Name: " + fullName);
	}
}
```

In this program, we declare two String variables `firstName` and `lastName`, and assign them the values "John" and "Doe", respectively. We then concatenate the two strings using the `+` operator, with a space in between them, and store the result in a variable called `fullName`. Finally, we print out the result using the `System.out.println()` method.

toString() Method:

```
public class Person {
	private String name;
	private int age;

	public Person(String name, int age) {
	this.name = name;
	this.age = age;
	}
```

```
	@Override
	public String toString() {
	return "Name: " + name + ", Age: " + age;
	}
}
public class ToStringMethodExercise {
	public static void main(String[] args) {
	Person person = new Person("John Doe", 30);
	System.out.println(person.toString());
	}
}
```

In this program, we create a custom class called `Person` that has two properties `name` and `age`, and a constructor that initializes them. We then override the `toString()` method of the class to return a string representation of the object, which includes the person's name and age.

String Conversion:

```
public class StringConversionExercise {
	public static void main(String[] args) {
	String numberString = "1234";
	int number = Integer.parseInt(numberString);
	System.out.println("Number: " + number);
	}
}
```

In this program, we declare a String variable called `numberString` and assign it the value "1234". We then convert the string to an int using the `Integer.parseInt()` method, and store the result in a variable called `number`. Finally, we print out the result using the `System.out.println()` method.

Character Extraction:

```
public class CharacterExtractionExercise {
	public static void main(String[] args) {
	String text = "Hello, world!";
	char firstCharacter = text.charAt(0);
	System.out.println("First Character: " + firstCharacter);
	}
}
```

In this program, we declare a String variable called `text` and assign it the value "Hello, world!". We then use the `charAt()` method to extract the first character of the string, which is the letter "H", and

store it in a variable called `firstCharacter`. Finally, we print out the character using the `System.out.println()` method.

String Comparison:

```
public class StringComparisonExercise {
      public static void main(String[] args) {
      String string1 = "Hello";
      String string2 = "hello";
      boolean isEqual = string1.equals(string2);
      System.out.println("Strings are equal: " + isEqual);
      }
}
```

In this program, we declare two String variables `string1` and `string2`, and assign them the values "Hello" and "hello", respectively. We then use the `equals()` method to compare the two strings, which returns `false` because the strings have different cases.

Finally, we print out the result using the `System.out.println()` method.

Searching:

```
public class SearchingExercise {
      public static void main(String[] args) {
      String text = "The quick brown fox jumps over the lazy
      dog."; int index = text.indexOf("brown");
      System.out.println("Index of 'brown': " + index);
     }
}
```

Replace:

```
public class ReplaceExercise {
      public static void main(String[] args) {
      String text = "The quick brown fox jumps over the lazy
      dog."; String modifiedText = text.replace("brown", "red");
      System.out.println("Modified Text: " + modifiedText);
      }
}
```

In this program, we declare a String variable called `text` and assign it the value "The quick brown fox jumps over the lazy dog.". We then use the `replace()` method to replace the substring "brown" with

"red" in the string, and store the modified string in a variable called `modifiedText`. Finally, we print out the modified string using the `System.out.println()` method.

Trim:

```
public class TrimExercise {
      public static void main(String[] args) {
      String text = " The quick brown fox jumps over the lazy dog. ";
      String modifiedText = text.trim();
      System.out.println("Modified Text: " + modifiedText);
      }
}
```

In this program, we declare a String variable called `text` and assign it the value " The quick brown fox jumps over the lazy dog. ", which contains leading and trailing whitespace. We then use the `trim()` method to remove the leading and trailing whitespace from the string, and store the modified string in a variable called `modifiedText`. Finally, we print out the modified string using the `System.out.println()` method.

StringBuffer:

```
public class StringBufferExercise {
      public static void main(String[] args) {
      StringBuffer stringBuffer = new StringBuffer("The quick brown
fox jumps over the lazy dog.");
      stringBuffer.append(" And so does the quick grey fox.");
      stringBuffer.insert(4, "lazy ");
      stringBuffer.delete(30, 34);
      String result = stringBuffer.toString();
      System.out.println(result);
      }
}
```

In this program, we declare a StringBuffer object called `stringBuffer` and initialize it with the value "The quick brown fox jumps over the lazy dog." using the constructor. We then use the `append()` method to add the string " And so does the quick grey fox." to the end of the StringBuffer object. We use the `insert()` method to insert the string "lazy " at index 4, which is before the word "quick". We then use the `delete()` method to delete the characters at indices 30 to 33, which is the word "grey". Finally, we convert the StringBuffer object to a String using the `toString()` method and print out the resulting string using the `System.out.println()` method.

Math Class:

The Math class in Java provides various methods for performing mathematical operations. Some of the commonly used methods are:

- `abs(int a)` - returns the absolute value of the integer value `a`.
- `abs(long a)` - returns the absolute value of the long value `a`.
- `abs(float a)` - returns the absolute value of the float value `a`.
- `abs(double a)` - returns the absolute value of the double value `a`.
- `ceil(double a)` - returns the smallest integer greater than or equal to the double value `a`.
- `floor(double a)` - returns the largest integer less than or equal to the double value `a`. • `max(int a, int b)` - returns the greater of the two int values `a` and `b`.
- `max(long a, long b)` - returns the greater of the two long values `a` and `b`. • `max(float a, float b)` - returns the greater of the two float values `a` and `b`. • `max(double a, double b)` - returns the greater of the two double values `a` and `b`. • `min(int a, int b)` - returns the smaller of the two int values `a` and `b`. • `min(long a, long b)` - returns the smaller of the two long values `a` and `b`. • `min(float a, float b)` - returns the smaller of the two float values `a` and `b`. • `min(double a, double b)` - returns the smaller of the two double values `a` and `b`. • `pow(double a, double b)` - returns the value of the double value `a` raised to the power of the double value `b`.
- `sqrt(double a)` - returns the square root of the double value `a`.

```
public class MathExercise {
      public static void main(String[] args) {
      int num1 = -5;
      double num2 = 3.75;
      double num3 = 9.3;
      double result1 = Math.abs(num1);
      double result2 = Math.ceil(num2);
      double result3 = Math.floor(num3);
      int result4 = Math.max(10, 20);
      int result5 = Math.min(10, 20);
      double result6 = Math.pow(2, 5);
      double result7 = Math.sqrt(25);
      System.out.println("Absolute value of " + num1 + " is " +
      result1); System.out.println("Ceiling of " + num2 + " is " +
      result2); System.out.println("Floor of " + num3 + " is " +
      result3);
      System.out.println("Max of 10 and 20 is " + result4);
      System.out.println("Min of 10 and 20 is " + result5);
      System.out.println("2 raised to the power of 5 is " +
      result6); System.out.println("Square root of 25 is " +
      result7);
      }
}
```

In this program, we use the Math class to perform various mathematical operations. We declare

variables `num1`, `num2`, and `num3` to store some numbers. We then use the Math methods to perform the required operations and store the results in variables `result1` through `result7`. Finally, we print out the results using the `System.out.println()` method.

Scanner Class:

The Scanner class in Java is used to read input from the user or from a file. It provides various methods for reading different types of input such as integers, doubles, strings, etc.

```
import java.util.Scanner;

public class ScannerExercise {
	public static void main(String[] args) {
	Scanner scanner = new Scanner(System.in);
	System.out.print("Enter a string: ");
	String strInput = scanner.next();
	System.out.print("Enter an integer: ");
	int intInput = scanner.nextInt();
	System.out.print("Enter a double: ");
	double doubleInput = scanner.nextDouble();
	System.out.println("String input: " + strInput);
	System.out.println("Integer input: " + intInput);
	System.out.println("Double input: " + doubleInput);
	scanner.close();
	}
}
```

ArrayList Class:

The ArrayList class in Java is used to store a collection of elements. It provides various methods for adding, removing, and accessing elements in the list.

```
import java.util.ArrayList;
import java.util.Arrays;

public class ArrayListExercise {
	public static void main(String[] args) {
	ArrayList<String> namesList = new ArrayList<String>();
	namesList.add("John");
	namesList.add("Mary");
	namesList.add("Peter");
	System.out.println("Size of the list: " +
	namesList.size()); System.out.println("Element at index 1:
	" + namesList.get(1));
```

```
        namesList.set(1, "David");
        System.out.println("Modified element at index 1: " +
namesList.get(1));
        namesList.remove(2);
        String[] namesArray = namesList.toArray(new String[0]);
        System.out.println("List: " + namesList);
        System.out.println("Array: " +
        Arrays.toString(namesArray)); }
}
```

In this program, we create an ArrayList object to store a list of names. We use the `add()` method to add some names to the list. We then use the `size()`, `get()`, `set()`, and `remove()` methods to perform various operations on the list. We use the `toArray()` method to convert the list to an array. Finally, we print out the list and the array using the `System.out.println()` method.

Methods of ArrayList Class:

- `add(Object element)`: Adds an element to the end of the list.
- `add(int index, Object element)`: Inserts an element at the specified index in the list.
- `get(int index)`: Returns the element at the specified index in the list.
- `set(int index, Object element)`: Replaces the element at the specified index in the list with the specified element.
- `remove(int index)`: Removes the element at the specified index in the list.
- `size()`: Returns the number of elements in the list.
- `toArray()`: Returns an array containing all of the elements in the list.
- `clear()`: Removes all elements from the list.

```
import java.util.ArrayList;

public class ArrayListDemo {
      public static void main(String[] args) {
      // create a new ArrayList
      ArrayList<String> list = new ArrayList<>();

      // add elements to the end of the list
      list.add("apple");
      list.add("banana");
      list.add("cherry");
      System.out.println("list after adding elements: " + list);

      // insert an element at the specified index
```

```
        list.add(1, "orange");
        System.out.println("list after inserting element: " + list);

        // get an element at the specified index
        String element = list.get(2);
        System.out.println("element at index 2: " + element);

        // replace an element at the specified index
        list.set(1, "pear");
        System.out.println("list after replacing element: " + list);

        // remove an element at the specified index
        list.remove(0);
        System.out.println("list after removing element: " + list);

        // get the number of elements in the list
        int size = list.size();
        System.out.println("number of elements in the list: " + size);

        // convert the list to an array
        String[] array = list.toArray(new String[0]);
      System.out.println("array from list: " + Arrays.toString(array));

        // clear the list
        list.clear();
        System.out.println("list after clearing: " + list);
        }
}
```

Exercises:

You work at a small business and you have been given a string of customer information that needs to be processed. The string contains the customer's name, email address, phone number, and mailing address, all separated by commas. Your task is to manipulate the string using Java's string methods to extract the customer's name, email address, and phone number, and format them into a new string.

Instructions

1. Start by prompting the user to input the customer information string. You can do this using Java's input() function.
2. Once you have the input string, use the split() method to split the string into an array of individual values. Since the values are separated by commas, you can use the string "," as the separator. Assign the resulting array to a variable.
3. Convert the array to an Arraylist and use the array indexing notation to extract the customer's

name, email address, and phone number from the list. Assign each value to a separate variable. 4. Use the replace() method to remove any spaces from the phone number string. 5. Use the format() method to create a new string that includes the customer's name, email address, and formatted phone number. The string should have the following format: "Customer Name: <name>\nEmail Address: <email>\nPhone Number: <phone>".

6. Finally, use the print() function to print out the new formatted string.

Example

Here's an example of what the program output might look like:

```
Please enter the customer information string: John Smith,
johnsmith@example.com, (123) 456-7890, 123 Main St
Customer Name: John Smith
Email Address: johnsmith@example.com
Phone Number: 1234567890
```

Tips

- Make sure to remove any extra whitespace from the name and email address strings using the strip() method.
- If the phone number string contains other characters besides digits and parentheses, you may need to use a combination of string methods (such as replace() and strip()) to remove them before formatting the string.
- Don't forget to include the newline character ("\n") in the format() string to create line breaks between the different pieces of information.