17TH JULY INTRODUCTION TO API

1. *Program to display current date and time in java*

Ans:

To display the current date and time in Java, you can use the java.util.Date and java.text.SimpleDateFormat classes. Here's a simple program that accomplishes this:

```
import java.util.Date;
import java.text.SimpleDateFormat;

public class CurrentDateTime {
    public static void main(String[] args) {
        // Create a Date object representing the current date and time
        Date currentDate = new Date();

        // Create a SimpleDateFormat object to format the date and time
        SimpleDateFormat dateFormat = new SimpleDateFormat("MM/dd/yyyy HH:mm:ss");

        // Format the current date and time to a string
        String formattedDateTime = dateFormat.format(currentDate);

        // Display the formatted date and time
        System.out.println("Current Date and Time: " + formattedDateTime);
    }
}
```

2. *Write a program to convert a date to a string in the format "MM/dd/yyyy".*

Ans:

To convert a date to a string in the "MM/dd/yyyy" format, you can use the java.text.SimpleDateFormat class. Here's a program that demonstrates how to do this:

```
java
Copy code
import java.util.Date;
import java.text.SimpleDateFormat;
```

```
public class DateToString {
    public static void main(String[] args) {
        // Create a Date object representing the desired date
        Date date = new Date(); // You can replace this with any other date

        // Create a SimpleDateFormat object to format the date
        SimpleDateFormat dateFormat = new SimpleDateFormat("MM/dd/yyyy");

        // Format the date to a string
        String formattedDate = dateFormat.format(date);

        // Display the formatted date
        System.out.println("Formatted Date: " + formattedDate);
    }
}
```

3. What is the difference between collections and streams? Explain with an Example.

Ans:

In Java, collections and streams are used for handling and processing groups of elements, but they have distinct characteristics:

Collections:

Collections are data structures used to store and manage groups of elements.

They are typically used to store elements in memory, like lists, sets, and maps.

Collections allow random access, which means you can access elements by their index.

Operations on collections are usually done sequentially, using loops like for-each or iterators.

Streams:

Streams are a way to process collections of data in a functional-style, supporting operations like filtering, mapping, and reducing.

They do not store data themselves; instead, they provide a pipeline to process data from a source, like a collection or an array.

Streams are designed to work with large datasets and can process elements in parallel to achieve better performance.

Operations on streams are usually chainable and result in a new stream or a final result.

Example:

Suppose you have a list of integers and you want to find the sum of the even numbers.

Using Collections:

java

Copy code

```
import java.util.*;

public class CollectionExample {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
        int sum = 0;
        for (int num : numbers) {
            if (num % 2 == 0) {
                sum += num;
            }
        }
        System.out.println("Sum of even numbers using Collections: " + sum);
    }
}
```

Using Streams:

java

Copy code

```
import java.util.*;
import java.util.stream.*;

public class StreamExample {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
```

```
        int sum = numbers.stream()
                .filter(num -> num % 2 == 0)
                .mapToInt(Integer::intValue)
                .sum();
        System.out.println("Sum of even numbers using Streams: " + sum);
    }
}
```

4. What is enums in java? explain with an example.

Ans:

In Java, enums (short for enumerations) are a special data type that allow you to define a set of named constants. Enumerations are used to represent fixed and predefined values, making the code more readable and maintainable.

Example:

java

Copy code

```
public class EnumExample {
    // Define an enum for days of the week
    enum Day {
        SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
    }

    public static void main(String[] args) {
        // Accessing enum constants
        Day today = Day.WEDNESDAY;

        // Switch statement with enum
        switch (today) {
            case MONDAY:
            case TUESDAY:
            case WEDNESDAY:
            case THURSDAY:
                System.out.println("Weekday");
```

```
            break;
        case FRIDAY:
            System.out.println("Friday");
            break;
        case SATURDAY:
        case SUNDAY:
            System.out.println("Weekend");
            break;
    }
  }
}
```

5. What are in built annotations in java?

Ans:

Java provides several built-in annotations that allow you to add metadata and instructions to your code for various purposes, such as compiler warnings, code generation, and runtime behaviors. Some common built-in annotations in Java are:

@Override: This annotation is used to indicate that a method in a subclass is intended to override a method in the superclass. If the method doesn't actually override any method from the superclass, the compiler will raise an error.

@Deprecated: This annotation is used to mark that a method, class, or field is deprecated and should no longer be used. When developers use deprecated elements, the compiler will generate a warning.

@SuppressWarnings: This annotation is used to suppress specific compiler warnings that might be generated in the code. It helps to avoid unnecessary warnings when you know certain conditions are met.

@FunctionalInterface: This annotation is used to specify that an interface is a functional interface, meaning it has only one abstract method. It is used for lambda expressions and method references.

@SafeVarargs: This annotation is used to suppress unchecked warnings that may arise when using varargs (variable-length arguments) with generics. It ensures the method doesn't perform unsafe operations with varargs.

These annotations are part of the Java standard library and provide valuable information to developers, tools, and the compiler to improve code quality and maintainability.