day-028-apis-and-anotations

1. Program to display current date and time in java
2. Write a program to convert a date to a string in the format "MM/dd/yyyy".
3. What is the difference between collections and streams? Explain with an Example.
4. What is enums in java? explain with an example.
5. What are in built annotations in java?

’1. Program to display current date and time in Java:

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class CurrentDateTimeExample {
    public static void main(String[] args) {
        LocalDateTime currentDateTime = LocalDateTime.now();
        DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");
        String formattedDateTime =
currentDateTime.format(formatter);
        System.out.println("Current date and time is: " +
formattedDateTime);
    }
}
```

This program uses the LocalDateTime class from the java.time package to get the current date and time. It then uses the DateTimeFormatter class to format the date and time into a string with the pattern "dd-MM-yyyy HH:mm:ss".

2. Program to convert a date to a string in the format "MMIddlyyyy":

```
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;

public class DateToStringExample {
    public static void main(String[] args) {
        LocalDate date = LocalDate.of(2023, 4, 12);
        DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("MM-dd-uuuu");
```

```
        String formattedDate = date.format(formatter);
        System.out.println("Formatted date is: " +
formattedDate);
    }
}
```

This program uses the LocalDate class from the java.time package to create a date object with the year, month, and day. It then uses the DateTimeFormatter class to format the date into a string with the pattern "MM-dd-yyyy". The resulting string is "0412-2023".

3. Difference between collections and streams:

Collections and streams are both used for processing collections of objects, but they have different characteristics and use cases.

Collections are data structures that hold a collection of elements, such as lists, sets, and maps. Collections provide methods for adding, removing, and accessing elements, and they can be used in loops or with iterators to perform operations on each element.

Streams are a way to process collections of elements in a functional style. Streams allow you to perform operations on each element in the collection using methods like map, filter, and reduce. Streams are designed to be lazy, which means that operations are only performed when needed, and they can be parallelized to take advantage of multi-core processors.

Here's an example that demonstrates the difference between using a collection and a stream to filter a list of strings:

```
import java.util.ArrayList;
import java.util.List;

public class CollectionVsStreamExample {
    public static void main(String[] args) {
        List<String> names = new ArrayList<>();
        names.add("Alice");
        names.add("Bob");
        names.add("Charlie");

        // Using a collection to filter names starting with
"A"
        List<String> filteredNames = new ArrayList<>();
```

```
        for (String name : names) {
            if (name.startsWith("A")) {
                filteredNames.add(name);
            }
        }
        System.out.println("Filtered names using collection: "
+ filteredNames);

        // Using a stream to filter names starting with "A"
        List<String> filteredNames2 = names.stream()
                .filter(name -> name.startsWith("A"))
                .toList();
        System.out.println("Filtered names using stream: " +
filteredNames2);
    }
}
```

In this example, we have a list of strings names that we want to filter to only include the strings that start with the letter "A". The first approach uses a collection to loop through each element in the list and add the matching elements to a new list. The second approach uses a stream to filter the list using a lambda expression that tests if the name starts with "A". The resulting filtered lists are the same, but the second approach using a stream is more concise and expressive.

4. Enums in Java:

Enums in Java are a special type of class that represents a fixed set of values. Enums are typically used to represent categories, states, or options that have a limited number of possible values. Enums provide type safety and are more expressive than using constants or integers to represent values.

Here's an example of how to define an enum in Java:

```
public enum DayOfWeek {
  MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY,
SUNDAY;
}
```

In this example, we define an enum called DayOfWeek that represents the days of the week. The enum has seven values, one for each day of the week.

We can use the enum in our code like this:

```
DayOfWeek day = DayOfWeek.MONDAY;
if (day == DayOfWeek.MONDAY || day == DayOfWeek.TUESDAY || day
== DayOfWeek.WEDNESDAY || day == DayOfWeek.THURSDAY || day ==
DayOfWeek.FRIDAY) {
    System.out.println("It's a weekday!");
} else {
    System.out.println("It's a weekend!");
}
```

In this example, we create a variable day of type DayOfWeek and assign it the value MONDAY. We then use an if statement to check if the day is a weekday (Monday through Friday). Because we're using an enum, we can use the enum values directly in our code, which makes the code more readable and less error-prone.

5. Built-in annotations in Java:

Java has several built-in annotations that provide additional information about classes, methods, and variables. Annotations are used to provide metadata that can be used by tools, frameworks, or other code to perform specific actions or make decisions.

Here are some examples of built-in annotations in Java:

@Override: This annotation is used to indicate that a method is intended to override a method in a superclass or interface. This annotation is optional but provides additional safety by generating a compile-time error if the method signature does not match the superclass or interface.

@Deprecated: This annotation is used to indicate that a class, method, or variable is deprecated and should not be used in new code. This annotation is typically used when a better alternative is available or when the feature is being phased out.

@FunctionalInterface: This annotation is used to indicate that an interface is a functional interface, which means that it has only one abstract method. This annotation is optional but provides additional safety by generating a compile-time error if the interface has more than one abstract method.