APIs and Annotation

1. Program to display current date and time in java

Ans1. **Using `LocalDateTime` (Java 8 and later)**

```
import java.time.LocalDateTime;

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

```
public class CurrentDateTime {

    public static void main(String[] args) {

        LocalDateTime now = LocalDateTime.now();

        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

        String formattedDateTime = now.format(formatter);



        System.out.println("Current Date and Time: " + formattedDateTime);
    }

}
```

Output (Example):

```
Current Date and Time: 2025-02-15 14:30:45
```

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

Ans2. Java program to convert a `LocalDate` to a string in the format `"MM/dd/yyyy"`:

Java Program:

```
import java.time.LocalDate;

import java.time.format.DateTimeFormatter;



public class DateToString {

    public static void main(String[] args) {

        // Get the current date
```

```
LocalDate date = LocalDate.now();

// Define the format "MM/dd/yyyy"

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("MM/dd/yyyy");

// Convert date to string

String formattedDate = date.format(formatter);
```

```
        // Display the formatted date

        System.out.println("Formatted Date: " + formattedDate);

    }

}
```

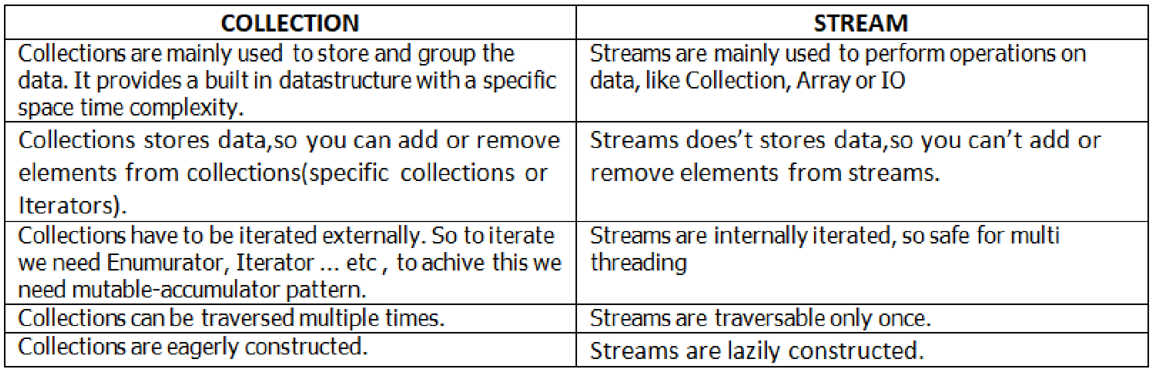
Example Output:

```
Formatted Date: 02/15/2025
```

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

Ans3.

Some basic differences are



COLLECTION	STREAM
Collections are mainly used to store and group the data. It provides a built in datastructure with a specific space time complexity.	Streams are mainly used to perform operations on data, like Collection, Array or IO
Collections stores data,so you can add or remove elements from collections(specific collections or Iterators).	Streams does't stores data,so you can't add or remove elements from streams.
Collections have to be iterated externally. So to iterate we need Enumurator, Iterator ... etc , to achive this we need mutable-accumulator pattern.	Streams are internally iterated, so safe for multi threading
Collections can be traversed multiple times.	Streams are traversable only once.
Collections are eagerly constructed.	Streams are lazily constructed.

Example: Collections vs. Streams

Using Collections (List)

import java.util.ArrayList;

import java.util.List;

```
public class CollectionsExample {

    public static void main(String[] args) {

        List<String> names = new ArrayList<>();

        names.add("Alice");

        names.add("Bob");

        names.add("Charlie");
        names.add("David");

        // Traditional Iteration

        for (String name : names) {

           if (name.startsWith("A")) {

              System.out.println(name);

           }
```

```
        }
    }
}
```

Output:

```
Alice
```

Using Streams

```
import java.util.Arrays;
import java.util.List;

import java.util.stream.Collectors;

public class StreamsExample {

   public static void main(String[] args) {

      List<String> names = Arrays.asList("Alice", "Bob", "Charlie", "David");
```

```
    // Using Stream API

    List<String> filteredNames = names.stream()

            .filter(name -> name.startsWith("A")) // Filter names starting with 'A'

            .collect(Collectors.toList()); // Collect results into a List

    System.out.println(filteredNames);

}
```

}

Output:

[Alice]

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

Ans4. An **enum** (short for enumeration) in Java is a special data type that represents a fixed set of constants. It is used when we have a set of predefined values that do not change, such as days of the week, months, or status codes.

Key Features of Enums:

- Defined using the `enum` keyword.
- Each enum constant is implicitly `public`, `static`, and `final`.
- Can have fields, constructors, and methods.
- Provides built-in methods like `values()` and `ordinal()`.

Example: Simple Enum

```
// Define an Enum for Days of the Week

enum Day {

    SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY;

}



public class EnumExample {
```

```
public static void main(String[] args) {

    // Using an Enum

    Day today = Day.FRIDAY;

    // Print Enum

    System.out.println("Today is: " + today);
}

    // Loop through Enum values

    System.out.println("All Days:");

    for (Day d : Day.values()) {

        System.out.println(d);

    }

  }

}
```

Output:

```
Today is: FRIDAY

All Days:

SUNDAY

MONDAY

TUESDAY

WEDNESDAY
```

THURSDAY

FRIDAY

SATURDAY

Example: Enum with Fields and Methods

// Define an Enum with Custom Fields and Methods

enum Size {

```
SMALL("S"), MEDIUM("M"), LARGE("L"), EXTRALARGE("XL");

private String abbreviation;

// Constructor

Size(String abbreviation) {

    this.abbreviation = abbreviation;
    }

    // Getter method

    public String getAbbreviation() {

        return abbreviation;

    }

}
```

```
}

public class EnumWithFields {

    public static void main(String[] args) {

        Size mySize = Size.LARGE;

        System.out.println("Selected Size: " + mySize);

        System.out.println("Size Abbreviation: " + mySize.getAbbreviation());

    }
```

}

Output:

```
Selected Size: LARGE

Size Abbreviation: L
```

5. What are in built annotations in java?

Ans5. **In-Built Annotations in Java**

Annotations in Java provide metadata about the code and help the compiler, tools, and frameworks process information efficiently. Java has several built-in annotations, categorized mainly as:

1. **Marker Annotations** (without parameters)
2. **Single-Value Annotations** (with one parameter)
3. **Multi-Value Annotations** (with multiple parameters)