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

Ans.

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

public class CurrentDateTime {
   public static void main(String[] args) {
      Date currentDate = new Date();


      SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy
HH:mm:ss");


      String formattedDateTime = dateFormat.format(currentDate);


      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.

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

public class DateToString {
   public static void main(String[] args) {

      Date date = new Date(); // You can replace this with any other Date object

      SimpleDateFormat dateFormat = new SimpleDateFormat("MM/dd/yyyy");

      String formattedDate = dateFormat.format(date);

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

3. **Ans.**	**What is the difference between collections and streams? Explain with an Example.** Collections and streams are both important concepts in Java for handling and processing data, but they serve different purposes and have different characteristics. Collections: • Collections in Java represent groups of objects, typically stored in data structures such as lists, sets, maps, etc. • They are used to store and manipulate data in memory. • Collections are concrete data structures that hold elements, and they provide methods for adding, removing, and accessing elements. • Collections are typically used for storing and managing finite sets of data. • Examples of collections in Java include ArrayList, LinkedList, HashSet, HashMap, etc. Streams: • Streams in Java represent sequences of elements that support aggregate operations on them. • They are not data structures like collections; instead, they are functional-style abstractions that allow you to process data in a declarative and pipeline-oriented manner. • Streams do not store data themselves; instead, they operate on data from a source, such as a collection, array, or I/O channel. • Streams support functional-style operations such as map, filter, reduce, etc., which allow you to perform complex data processing tasks in a concise and efficient manner. • Streams are typically used for processing large or infinite streams of data and performing operations such as filtering, mapping, sorting, and aggregating. • Streams are lazy; they do not perform any computation until a terminal operation is invoked on them.
4. **Ans.**	**What is enum in java? Explain with example** In Java, an enum, short for enumeration, is a special data type that represents a fixed set of constants. Enums provide a way to define a group of related

constants in a more structured and type-safe manner. Each constant in an enum is considered an instance of the enum type.

```
// Define an enum called Day representing the days of the week
enum Day {
   SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
}

public class EnumExample {
   public static void main(String[] args) {
      // Access enum constants
      Day today = Day.MONDAY;

      // Switch statement with enum
      switch (today) {
         case MONDAY:
            System.out.println("Today is Monday");
            break;
         case TUESDAY:
            System.out.println("Today is Tuesday");
            break;
         case WEDNESDAY:
            System.out.println("Today is Wednesday");
            break;
         // Add cases for other days as needed
         default:
            System.out.println("Today is not Monday, Tuesday, or Wednesday");
      }

      // Iterate over enum constants
      System.out.println("All days of the week:");
      for (Day day : Day.values()) {
         System.out.println(day);
      }
   }
}
```

Explanation:

- In this example, we define an enum called Day, which represents the days of the week.
- The enum constants (SUNDAY, MONDAY, TUESDAY, etc.) are declared within the enum definition.
- Enums are implicitly final and static, so they cannot be subclassed or instantiated using the new keyword.

- We can use enum constants just like any other constants in Java.
- We can use enums in switch statements for easy and type-safe branching based on enum values.
- We can iterate over all enum constants using the values() method, which returns an array of all enum constants.

5. **What are in built annotations in java?**

Ans.

Built-in annotations in Java are predefined annotations provided by the Java language. They are part of the Java Standard Library and are used for various purposes such as providing metadata about the code, instructing the compiler, or influencing runtime behavior. Here are some commonly used built-in annotations in Java:

1. **@Override**
    - Indicates that a method is intended to override a method in a superclass.
    - Helps the compiler catch errors if the method does not actually override a method in the superclass.

@Deprecated

- Marks a class, method, or field as deprecated, indicating that it should no longer be used.
- The compiler generates a warning when the deprecated element is used