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

```
import java.time.*;
public class Demo {
   public static void main(String[] args) {
      LocalDate dt = LocalDate.now();
      System.out.println(dt);
      int day = dt.getDayOfMonth();
      int month = dt.getMonthValue();
      int year = dt.getYear();

      System.out.println(day + "/" + month + "/" + year);

      LocalTime time = LocalTime.now();
      System.out.println(time);
      int hour = time.getHour();
      int min = time.getMinute();
      int sec = time.getSecond();
      int nano = time.getNano();

      System.out.println(hour + ":" + min + ":" + sec + ":" + nano);
   }
}
```

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

```
import java.time.*;

public class Demo {
   public static void main(String[] args) {
      LocalDate dt = LocalDate.now();
      System.out.println(dt);
      int day = dt.getDayOfMonth();
      int month = dt.getMonthValue();
      int year = dt.getYear();

      System.out.println(day + "/" + month + "/" + year);

      LocalTime time = LocalTime.now();
      System.out.println(time);
      int hour = time.getHour();
      int min = time.getMinute();
      int sec = time.getSecond();
      int nano = time.getNano();

      System.out.println(hour + ":" + min + ":" + sec + ":" + nano);
   }
}
```

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

Collections are used to store and manipulate groups of objects. They are part of the *java.util* package and include various data structures like *List*, *Set*, and *Map*. Collections are typically mutable, meaning you can add, remove, or update elements.

Streams, introduced in Java 8, provide a functional approach to processing sequences of elements. They allow you to perform operations like filtering, mapping, and reducing in a declarative manner. Streams are immutable and support lazy evaluation, meaning operations are only performed when necessary.

Example:

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

public class CollectionsVsStreams {
    public static void main(String[] args) {
        List<String> companyList = new ArrayList<>();
        companyList.add("Google");
        companyList.add("Apple");
        companyList.add("Microsoft");

        // Using Collections
        Collections.sort(companyList);
        for (String company : companyList) {
            System.out.println(company);
        }

        // Using Streams
        companyList.stream()
                .sorted()
                .forEach(System.out::println);
    }
}
```

Output:

```
Apple
Google
Microsoft
```

4. What is enums in Java? Explain with an example.

Enums in Java are a special type of class that represents a group of constants (unchangeable variables). Enums are used to define a collection of constants that can be used to represent a fixed set of related values.

Example:

```
public class EnumExample {
  public enum Day {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
  }

  public static void main(String[] args) {
    Day today = Day.FRIDAY;

    switch (today) {
      case MONDAY:
        System.out.println("Mondays are tough.");
        break;
      case FRIDAY:
        System.out.println("Fridays are great!");
        break;
      default:
        System.out.println("Midweek days are so-so.");
        break;
    }
  }
}
```

Output:

```
Fridays are great!
```

5. What are in-built annotations in Java?

In-built annotations in Java are predefined annotations provided by the Java language. They are used to provide metadata for Java code and can be used to influence the behavior of the compiler or runtime.

Common In-built Annotations:

- *@Override*: Indicates that a method is intended to override a method in a superclass.
- *@Deprecated*: Marks a method, class, or field as deprecated, meaning it should no longer be used.
- *@SuppressWarnings*: Instructs the compiler to suppress specific warnings for the annotated element.
- *@FunctionalInterface*: Indicates that an interface is intended to be a functional interface, which has exactly one abstract method.