Java Project Assignment

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

Ans:

```
package API;
import java.time.*;
public class DateTime {
   public static void main(String[] args) {
      LocalDate date = LocalDate.now();
      System.out.println(date);
      LocalTime time=LocalTime.now();
      System.out.println(time);
   }
}
```

Output:

2023-08-10
19:13:58.798872300

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

Ans:

```
package API;

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

public class DateToString2 {
   public static void main(String[] args) {

      LocalDate date = LocalDate.of(2023, 8, 10);
      DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("MM/dd/yyyy");

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

```
    }
}
```

Output:

Formatted Date: 08/10/2023

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

Ans:

STREAMS	COLLECTIONS
It doesn't store data, it operates on the source data structure i.e collection.	It stores/holds all the data that the data structure currently has in a particular data structure like Set, List or Map,
They use functional interfaces like lambda which makes it a good fit for programming language.	They don't use functional interfaces.
Java Streams are consumable i.e; to traverse the stream, it needs to be created every time.	They are non-consumable i.e; can be traversable multiple times without creating it again.
Java streams support both sequential and parallel processing.	It supports parallel processing and parallel processing can be very helpful in achieving high performance.
Streams are iterated internally by just mentioning the operations.	Collections are iterated externally using loops.

Example: Collections

```
import java.util.*;
import java.util.Comparator;
import java.util.List;

public class CollectionEx {
   public static void main(String[] args) {
      List<String> list = new ArrayList<>();
      list.add("Google");
      list.add("Apple");
      list.add("Microsoft");
      list.add("Facebook");
      Comparator<String> com = (String o1, String o2) -> o1.compareTo(o2);
      Collections.sort(list);
      for(String name:list){
         System.out.println(name);
      }
   }
}
```

Output:

Apple
Facebook
Google
Microsoft

Example: Streams

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

public class Streams {
   public static void main(String[] args)
   {
      List<String> CompanyList = new ArrayList<>();
      CompanyList.add("Google");
      CompanyList.add("Apple");
      CompanyList.add("Microsoft");
      CompanyList.stream().sorted().forEach(
```

```
            System.out::println);
    }
}
```

Output:

Apple

Google

Microsoft

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

Ans: We can use enum to define a group of named constants. Enums are used to represent a collection of related constants that have a common purpose. Each constant in an enum is an instance of the enum type, and they are typically defined as public static final fields.
Here's an example of how to define an enum in Java:

Example:

```
package API;

public class EnumEx {
    public enum DayOfWeek {
        MONDAY,
        TUESDAY,
        WEDNESDAY,
        THURSDAY,
        FRIDAY,
        SATURDAY,
        SUNDAY
    }
    public static void main(String args[]){
        for(DayOfWeek d:DayOfWeek.values())
            System.out.println(d);
    }
}
```

Here we define an enum called "DayOfWeek" that represents the days the week. The enum has seven constants, each representing a day of the week. The constants are defined in all uppercase letters by convention.

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

Ans: built-in annotations in Java:

@Override
@Deprecated
@SuppressWarnings
@FunctionalInterface
@Retention
@Target
@Documented
@Inherited

These built-in annotations in Java are used to provide additional information to the Java compiler and other tools. They help improve code readability, maintainability, and safety by enforcing specific rules and behaviours in Java code.