**1. Program to Display Current Date and Time in Java**

To display the current date and time in Java, you can use the LocalDateTime class from the java.time package, which is available from Java 8 onwards.

Example:

java

CopyEdit

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

public class CurrentDateTime {

public static void main(String[] args) {

// Get the current date and time

LocalDateTime now = LocalDateTime.now();

// Format the date and time

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

// Print the formatted date and time

System.out.println("Current Date and Time: " + now.format(formatter));

}

}

This program displays the current date and time in the yyyy-MM-dd HH:mm:ss format.

**2. Write a Program to Convert a Date to a String in the Format "MM/dd/yyyy"**

To convert a date into a string with a specific format, you can use the DateTimeFormatter class with LocalDate or Date in Java.

Example:

java

CopyEdit

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 date format

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

// Convert the date to a string

String formattedDate = date.format(formatter);

// Print the formatted date

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

}

}

This program converts the current date into the MM/dd/yyyy format.

**3. What is the Difference Between Collections and Streams? Explain with an Example**

* **Collections**:  
  Collections are a framework that provides methods to store, retrieve, and manipulate data in different types of data structures like lists, sets, queues, etc. They operate on the data directly and allow modification of the data structure.
* **Streams**:  
  Streams are introduced in Java 8 and provide a functional approach to process sequences of elements. Streams are not data structures themselves but are used to perform operations like filtering, mapping, and reducing on collections or arrays.

**Key Differences**:

* **Mutability**: Collections are mutable, while streams are immutable.
* **Execution**: Collections work eagerly (all operations are executed immediately), while streams support lazy evaluation (operations are executed only when a terminal operation is invoked).
* **API**: Collections use imperative code (looping), while streams use declarative code (using lambdas).

**Example**:

java

CopyEdit

import java.util.\*;

import java.util.stream.\*;

public class CollectionVsStream {

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

// Using Collections (imperative)

List<Integer> doubledNumbers = new ArrayList<>();

for (Integer num : numbers) {

doubledNumbers.add(num \* 2);

}

System.out.println("Using Collections: " + doubledNumbers);

// Using Streams (declarative)

List<Integer> doubledStream = numbers.stream()

.map(num -> num \* 2)

.collect(Collectors.toList());

System.out.println("Using Streams: " + doubledStream);

}

}

In this example, both collections and streams perform the same operation (doubling the numbers), but streams provide a more concise and functional approach.

**4. What is Enums in Java? Explain with an Example**

**Enums** in Java are special types that represent a group of constants (unchangeable variables). They are used to represent a fixed set of related constants such as days of the week, months, or directions. Enums are type-safe, meaning they restrict the variable to have only the defined constants.

Example:

java

CopyEdit

enum Day {

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

}

public class EnumExample {

public static void main(String[] args) {

Day day = Day.MONDAY;

switch(day) {

case MONDAY:

System.out.println("Start of the week");

break;

case FRIDAY:

System.out.println("End of the week");

break;

default:

System.out.println("Midweek day");

}

}

}

In this example, the Day enum defines the days of the week, and the switch statement uses enum values to determine the day of the week.

**5. What are Built-in Annotations in Java?**

Java provides several built-in annotations that serve different purposes. Some of the commonly used ones are:

* **@Override**: Indicates that a method is overriding a method from a superclass.
* **@Deprecated**: Marks a method or class as deprecated, meaning it should not be used anymore.
* **@SuppressWarnings**: Tells the compiler to suppress specific warnings, such as unchecked warnings or deprecation warnings.
* **@FunctionalInterface**: Marks an interface as a functional interface, meaning it has exactly one abstract method.
* **@SafeVarargs**: Suppresses warnings related to varargs parameters.

Example:

java

CopyEdit

public class AnnotationExample {

@Override

public String toString() {

return "This is an example";

}

@Deprecated

public void oldMethod() {

System.out.println("This method is outdated");

}

@SuppressWarnings("unchecked")

public void useUncheckedCode() {

List list = new ArrayList();

list.add("Item");

}

}

**1. Program to Display Current Date and Time in Java**

To display the current date and time in Java, you can use the LocalDateTime class from the java.time package, which is available from Java 8 onwards.

Example:

java

CopyEdit

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

public class CurrentDateTime {

public static void main(String[] args) {

// Get the current date and time

LocalDateTime now = LocalDateTime.now();

// Format the date and time

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

// Print the formatted date and time

System.out.println("Current Date and Time: " + now.format(formatter));

}

}

This program displays the current date and time in the yyyy-MM-dd HH:mm:ss format.

**2. Write a Program to Convert a Date to a String in the Format "MM/dd/yyyy"**

To convert a date into a string with a specific format, you can use the DateTimeFormatter class with LocalDate or Date in Java.

Example:

java

CopyEdit

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 date format

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

// Convert the date to a string

String formattedDate = date.format(formatter);

// Print the formatted date

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

}

}

This program converts the current date into the MM/dd/yyyy format.

**3. What is the Difference Between Collections and Streams? Explain with an Example**

* **Collections**:  
  Collections are a framework that provides methods to store, retrieve, and manipulate data in different types of data structures like lists, sets, queues, etc. They operate on the data directly and allow modification of the data structure.
* **Streams**:  
  Streams are introduced in Java 8 and provide a functional approach to process sequences of elements. Streams are not data structures themselves but are used to perform operations like filtering, mapping, and reducing on collections or arrays.

**Key Differences**:

* **Mutability**: Collections are mutable, while streams are immutable.
* **Execution**: Collections work eagerly (all operations are executed immediately), while streams support lazy evaluation (operations are executed only when a terminal operation is invoked).
* **API**: Collections use imperative code (looping), while streams use declarative code (using lambdas).

**Example**:

java

CopyEdit

import java.util.\*;

import java.util.stream.\*;

public class CollectionVsStream {

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

// Using Collections (imperative)

List<Integer> doubledNumbers = new ArrayList<>();

for (Integer num : numbers) {

doubledNumbers.add(num \* 2);

}

System.out.println("Using Collections: " + doubledNumbers);

// Using Streams (declarative)

List<Integer> doubledStream = numbers.stream()

.map(num -> num \* 2)

.collect(Collectors.toList());

System.out.println("Using Streams: " + doubledStream);

}

}

In this example, both collections and streams perform the same operation (doubling the numbers), but streams provide a more concise and functional approach.

**4. What is Enums in Java? Explain with an Example**

**Enums** in Java are special types that represent a group of constants (unchangeable variables). They are used to represent a fixed set of related constants such as days of the week, months, or directions. Enums are type-safe, meaning they restrict the variable to have only the defined constants.

Example:

java

CopyEdit

enum Day {

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

}

public class EnumExample {

public static void main(String[] args) {

Day day = Day.MONDAY;

switch(day) {

case MONDAY:

System.out.println("Start of the week");

break;

case FRIDAY:

System.out.println("End of the week");

break;

default:

System.out.println("Midweek day");

}

}

}

In this example, the Day enum defines the days of the week, and the switch statement uses enum values to determine the day of the week.

**5. What are Built-in Annotations in Java?**

Java provides several built-in annotations that serve different purposes. Some of the commonly used ones are:

* **@Override**: Indicates that a method is overriding a method from a superclass.
* **@Deprecated**: Marks a method or class as deprecated, meaning it should not be used anymore.
* **@SuppressWarnings**: Tells the compiler to suppress specific warnings, such as unchecked warnings or deprecation warnings.
* **@FunctionalInterface**: Marks an interface as a functional interface, meaning it has exactly one abstract method.
* **@SafeVarargs**: Suppresses warnings related to varargs parameters.

Example:

java

CopyEdit

public class AnnotationExample {

@Override

public String toString() {

return "This is an example";

}

@Deprecated

public void oldMethod() {

System.out.println("This method is outdated");

}

@SuppressWarnings("unchecked")

public void useUncheckedCode() {

List list = new ArrayList();

list.add("Item");

}

}