1. Program to display current date and time in Java?

**Ans:**

// Import required packages

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

public class CurrentDateTimeExample {

  public static void main(String[] args) {

    // Get current date and time

    LocalDateTime current = LocalDateTime.now();

    System.out.println("Current Date and Time is: " + current);

    // Format date and time

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

    String formatted = current.format(format);

    System.out.println("Formatted Date and Time is: " + formatted);

  }

}

1. Write a program to convert a date to a string in the format “MM/DD/YYYY”.

**Ans:**

// Import required packages

import java.text.SimpleDateFormat;

import java.util.Date;

public class DateToStringExample {

  public static void main(String[] args) {

    // Create a Date object

    Date date = new Date();

    System.out.println("Date object is: " + date);

    // Create a SimpleDateFormat object with the desired pattern

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

    // Convert the Date object to a String object using the format() method

    String formatted = format.format(date);

    System.out.println("Formatted date is: " + formatted);

  }

}

1. What is the difference between collections and streams? Explain with an example.

**Ans:** Collections and streams are two different ways of working with data in Java. Collections are used to store and group the data in a particular data structure like List, Set, or Map. [Streams are used to perform complex data processing operations like filtering, matching, mapping, etc on stored data such as arrays, collections, or I/O resources1](https://www.javaguides.net/2021/12/collections-vs-streams-in-java.html).

Some of the main differences between collections and streams are:

* Collections are eagerly constructed, meaning all the elements are computed and stored in memory at once. [Streams are lazily constructed, meaning the elements are computed and consumed only when needed](about:blank)[2](https://www.appsdeveloperblog.com/streams-vs-collections-in-java/)[3](https://javaconceptoftheday.com/collections-and-streams-in-java/).
* Collections can be modified by adding or removing elements. [Streams are immutable, meaning they cannot be changed once created](about:blank)[1](https://www.javaguides.net/2021/12/collections-vs-streams-in-java.html)[3](https://javaconceptoftheday.com/collections-and-streams-in-java/).
* Collections can be traversed multiple times by using an iterator or a for-each loop. [Streams can be traversed only once by using a terminal operation1](https://www.javaguides.net/2021/12/collections-vs-streams-in-java.html)[3](https://javaconceptoftheday.com/collections-and-streams-in-java/).
* Collections perform external iteration, meaning the user has to explicitly iterate over the elements. [Streams perform internal iteration, meaning the iteration is done by the stream behind the scenes1](https://www.javaguides.net/2021/12/collections-vs-streams-in-java.html)[3](https://javaconceptoftheday.com/collections-and-streams-in-java/).

Here is an example of using a collection and a stream to filter out even numbers from a list of integers:

// Using a collection

List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6);

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

for (Integer num : list) {

if (num % 2 == 0) {

even.add(num);

}

}

System.out.println(even); // [2, 4, 6]

// Using a stream

List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6);

List<Integer> even = list.stream()

.filter(num -> num % 2 == 0)

.collect(Collectors.toList());

System.out.println(even); // [2, 4, 6]

1. What is enums in java? Explain with an example.

**Ans:** [Enums are a special type of class in Java that represent a **group of named constants** that are known at compile time1](https://www.geeksforgeeks.org/enum-in-java/)[2](https://www.javatpoint.com/enum-in-java)[3](https://www.w3schools.com/java/java_enums.asp). [Enums are useful when you want to define a fixed set of values for a variable, such as days of the week, seasons, colors, directions, etc](about:blank)[2](https://www.javatpoint.com/enum-in-java)[3](https://www.w3schools.com/java/java_enums.asp).

Some of the features of enums are:

* [Enums **internally inherit** the Enum class, which provides some common methods and functionality1](https://www.geeksforgeeks.org/enum-in-java/)[2](https://www.javatpoint.com/enum-in-java).
* [Enums can **implement interfaces** but cannot **inherit other classes**](about:blank)[1](https://www.geeksforgeeks.org/enum-in-java/)[2](https://www.javatpoint.com/enum-in-java).
* [Enums can have **constructors**, **fields**, **methods**, and **inner classes**](about:blank)[1](https://www.geeksforgeeks.org/enum-in-java/)[2](https://www.javatpoint.com/enum-in-java).
* [Enums can be used as **arguments** for switch statements](about:blank)[1](https://www.geeksforgeeks.org/enum-in-java/)[3](https://www.w3schools.com/java/java_enums.asp).
* [Every enum constant is implicitly **public static final**](about:blank)[1](https://www.geeksforgeeks.org/enum-in-java/).

Here is an example of defining and using an enum in Java:

// Define an enum outside a class

enum Season {

SPRING, SUMMER, AUTUMN, WINTER;

}

public class EnumExample {

public static void main(String[] args) {

// Declare an enum variable

Season season = Season.SUMMER;

System.out.println("Current season is: " + season);

// Use an enum in a switch statement

switch (season) {

case SPRING:

System.out.println("It is spring time!");

break;

case SUMMER:

System.out.println("It is summer time!");

break;

case AUTUMN:

System.out.println("It is autumn time!");

break;

case WINTER:

System.out.println("It is winter time!");

break;

default:

System.out.println("Invalid season!");

}

}

}

Output:

Current season is: SUMMER

It is summer time!

1. What are in built annotations in java?

**Ans:** Annotations are a way of adding **metadata** to Java code, such as classes, methods, variables, parameters, etc. [Annotations can provide information to the compiler, the runtime, or other tools about how to process the code1](https://www.geeksforgeeks.org/annotations-in-java/)[2](https://www.w3schools.blog/annotations-java)[3](https://www.javatpoint.com/java-annotation).

Java has some **built-in annotations** that are predefined and have a specific meaning and purpose. [They can be divided into two categories: **standard** and **meta**1](https://www.geeksforgeeks.org/annotations-in-java/)[4](https://www.baeldung.com/java-default-annotations).

The **standard annotations** are:

* [@Deprecated: Indicates that a class, method, field, or parameter is no longer recommended and may be removed in the future](about:blank)[1](https://www.geeksforgeeks.org/annotations-in-java/)[2](https://www.w3schools.blog/annotations-java)[3](https://www.javatpoint.com/java-annotation).
* [@Override: Indicates that a method overrides a method declaration in a superclass or interface](about:blank)[1](https://www.geeksforgeeks.org/annotations-in-java/)[2](https://www.w3schools.blog/annotations-java)[3](https://www.javatpoint.com/java-annotation).
* [@SuppressWarnings: Indicates that the compiler should ignore one or more warnings for the annotated element](about:blank)[1](https://www.geeksforgeeks.org/annotations-in-java/)[2](https://www.w3schools.blog/annotations-java)[3](https://www.javatpoint.com/java-annotation).
* [@FunctionalInterface: Indicates that an interface is intended to be a functional interface, meaning it has only one abstract method](about:blank)[4](https://www.baeldung.com/java-default-annotations).
* [@Native: Indicates that a field is a constant that has a native implementation in the native code4](https://www.baeldung.com/java-default-annotations).

The **meta annotations** are:

* [@Target: Indicates the kinds of elements that an annotation type can be applied to, such as classes, methods, fields, etc](about:blank)[1](https://www.geeksforgeeks.org/annotations-in-java/)[4](https://www.baeldung.com/java-default-annotations).
* [@Retention: Indicates how long an annotation type should be retained by the compiler or the runtime, such as source level, class level, or runtime level](about:blank)[1](https://www.geeksforgeeks.org/annotations-in-java/)[4](https://www.baeldung.com/java-default-annotations).
* [@Documented: Indicates that an annotation type should be included in the generated documentation for the annotated element](about:blank)[1](https://www.geeksforgeeks.org/annotations-in-java/)[4](https://www.baeldung.com/java-default-annotations).
* [@Inherited: Indicates that an annotation type is automatically inherited by subclasses of the annotated class](about:blank)[1](https://www.geeksforgeeks.org/annotations-in-java/)[4](https://www.baeldung.com/java-default-annotations).

Here is an example of using some of the built-in annotations in Java:

// Define a custom annotation using meta annotations

@Documented

@Target(ElementType.METHOD)

@Retention(RetentionPolicy.RUNTIME)

@interface CustomAnnotation {

String value() default "Hello";

}

// Define an interface using @FunctionalInterface

@FunctionalInterface

interface MyInterface {

void doSomething();

}

// Define a class using @Deprecated and @Override

class MyClass implements MyInterface {

@Deprecated

public void oldMethod() {

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

}

@Override

public void doSomething() {

System.out.println("This method overrides the interface method");

}

}

// Define another class using @SuppressWarnings and @CustomAnnotation

@SuppressWarnings("deprecation")

class MyOtherClass extends MyClass {

@CustomAnnotation("World")

public void newMethod() {

System.out.println("This method has a custom annotation");

oldMethod(); // No warning because of @SuppressWarnings

}

}

I hope this gives you an overview of the built-in annotations in Java. Do you have any other questions about Java programming?