DAY-20

# JAVA:

## DATE & TIME IN JDK:

FUNCTIONS:

### LOCAL DATE:

* It belongs to java.time package.
* It is used to represent data without time and time zone.
* Eg: 2025-05-05

### LOCAL TIME:

* It is used to represent only Time- like 10:45:30. It is in 24-hour format only.
* Eg: 10:45:30

### LOCAL DATE TIME:

* It is used to represent a date and time together. But without a time-zone.
* Eg: 2025-05-26 T12:28:40

### ZONE DATE TIME:

* It is used to represent a complete date and time with a time zone.
* Eg: 202-05-26 T15:30:00 +05:30 [Asia/Kolkata]

### INSTANT-UTC TIMESTAMP:

* Instant is a class in java.time.package.
* It represents a specific moment on the timeline in UTC (Co-ordinated Universal time).
* It is typically used for timestamps-like when a file was created, or a log entry was made.
* import java.time.Instant;

### DURATION-TIME DIFFERENCE:

* Duration is a class in java.time.package.
* It is used to represent the amoun t of time between two temporal objects, typically Local
* Time, LocalDateTime, or Instant

### PERIOD-DATE DIFFERENCE:

* Period is a class in the java.time.package.
* It is used to represent a date-based amount of time like “2 years, 3 months, and 19 days”.
* It is used to calculate the difference between two LocalDate values.

## ADVANCES DATE-TIME FUNCTIONS:

1. ZONEID
2. ZONEOFFSET
3. OFFSETDATETIME
4. CLOCK
5. TEMPORAL ADJUSTERS
6. DATETIME FORMATTER
7. CHRONO UNIT

## REFLECTION API:

* Java Reflection is a process of examining or modifying the run-time behavior of a class at run-time.

or

* It is commonly used by programs which require the ability to examine or modify the run-time behavior of applications running in JVM.
* Java.lang and java.lang.reflect packages provide classes for java reflection.
* It is mainly used in
  + IDE
  + Debugger
  + Test tools
* Java.lang.Class class performs many tasks: -
  + Provides methods to get metadata of a class at run-time.
  + Provides methods to examine and change the run-time behavior of a class
* To get the object of Class class:
  + forName() method of Class class
  + getClass() method of Class class
  + the.class syntax

## NEW INSTANCE:

* The new Instance () method of Class class and Constructor class is used to create a new Instance.
* The new Instance () method of a Class class can invoke a zero-argument constructor. Whereas the new Instance () method of constructor class can invoke any number of arguments.
* So, constructor class is preferred over Class class.

## JAVAP TOOL:

* javap command dis assembles a class file.
* The javap command displays information about the fields, constructors, and methods present in a class file.
* Syntax:

Javap fully\_Class\_Name.

## ANNOTATIONS:

Annotations is a tag that represents the metadata (information about the data (or) data about the data) attached with class, interface methods.

In java they provide additional information. And it is an alternative to XML and Java Maker Interfaces.

BUILT-IN JAVA ANNOTATIONS:

1. @OVERRIDE:

This Annotation defines the subclass method, and it overrides the parent-class method. If it is not there, then a compile-time error occurs.

1. @SUPPRESSWARNINGS:

It is used to suppress warnings issued by compiler

1. @DPRECATCH:

It is deprecated, so the compiler prints a warning. It informs the user that it may be removed in future versions.

JAVA CUSTOM ANNOTATIONS:

The @interface element is used to declare an annotation.

Eg: @interface MyAnnotation

{

}

RULES FOR CUSTOM ANNOTATION:

* The method should not have any throws clauses.
* Method should return one of the following:
* Primitive- data types.
* String
* Class
* Enum or array

TYPES OF ANNOTATIONS:

1. Marker Annotation - an annotation that has no method
2. Single-value Annotation - that has one method
3. Multi-value Annotation – that has more than one method

CODE FOR MULTI-VALUE ANNOTATION:

@MyAnnotation (value1=10, value2=”Simmu”, value= “NANI”)

# CLEAN CODE:

## GETTING STARTED:

* Clean code means writing code that is easy to read, understand, and maintain.
* It focuses on simplicity.
* Avoid unnecessary complexity.
* It should be fun to write.
* It should follow common best practices and patterns.
* It should avoid unintuitive names, complex nesting, and big code blocks.

## NAMING: VARIABLES, FUNCTIONS, CLASSES & MORE

Use meaningful and descriptive names.

Follow naming conventions (snake case, camel case, Pascal case, kebab case)

Avoid abbreviations until they are understood.

Examples:

* userEmail () instead of ue ()
* calculateTotal instead of calc ()