? PySpark-Based Low-Level Design (LLD): Temperature Converter System

---

? Difficulty Level: Easy | Total Marks: 10

Standards Followed: 4 Functions | 3 Visible Test Cases | 2 Hidden Test Cases

---

? Concepts Tested

- ? Loading lists into Spark DataFrames
- ? Column transformation using expressions
- ? Aggregation functions (e.g. `avg`)
- ? Conditional logic with `when` / `otherwise`
- ? Class-based method organization

? Problem Statement

Design a PySpark-based system to manage Celsius temperatures, supporting operations like:

- Loading a list of temperatures
- Converting to Fahrenheit
- Calculating average temperature
- Categorizing as `"Cold"`, `"Moderate"`, or `"Hot"`

? Operations

---

1. Load Temperatures

Function Prototype:

```
def load_temperatures(self, temp_list: list) -> DataFrame:
```

- ? Input: List of float/int values (e.g., `[10, 20, 30]`)

- □ Output: Spark DataFrame with column `celsius`

---

2. Convert to Fahrenheit

Function Prototype:

```
def convert_to_fahrenheit(self, df: DataFrame) -> DataFrame:
```

- □ Input: DataFrame with `celsius`
- □ Output: DataFrame with new column `fahrenheit`

---

3. Calculate Average Temperature

Function Prototype:

```
def calculate_average(self, df: DataFrame) -> float:
```

- □ Input: DataFrame with `celsius`
- □ Output: Float value (average of column)

---

4. Categorize Temperatures

Function Prototype:

```
def categorize_temperatures(self, df: DataFrame) -> DataFrame:
```

- □ Input: DataFrame with `celsius`
- □ Output: DataFrame with column `category` (values: `"Cold"`, `"Moderate"`, `"Hot"`)

□ Implementation Code (`solution.py`)

```
from pyspark.sql import SparkSession, DataFrame
from pyspark.sql.functions import col, when, round, avg
```

```
class TemperatureConverter:
    def __init__(self, spark: SparkSession):
        self.spark = spark

    def load_temperatures(self, temp_list: list) -> DataFrame:
        """
        Convert a list of numbers into a DataFrame with column 'celsius'.
        Hint: Use self.spark.createDataFrame(...)
        """
        pass  # TODO

    def convert_to_fahrenheit(self, df: DataFrame) -> DataFrame:
        """
        Add a 'fahrenheit' column using formula (celsius * 9/5) + 32.
        Hint: Use withColumn and round()
        """
        pass  # TODO

    def calculate_average(self, df: DataFrame) -> float:
        """
        Compute average of celsius column.
        Hint: Use df.agg(...) and .first()[0]
        """
        pass  # TODO

    def categorize_temperatures(self, df: DataFrame) -> DataFrame:
        """
        Add a column 'category' based on value of celsius.
        Hint: Use when().otherwise() for categorization
        """
        pass  # TODO
```

□ Test Case Table

Test Case ID	Description	Function(s)	Marks
TC1	Load temperatures	`load_temperatures()`	2
TC2	Convert to Fahrenheit	`convert_to_fahrenheit()`	2
TC3	Calculate average	`calculate_average()`	2
HTC1	Categorize edge values	`categorize_temperatures()`	2
HTC2	Handle only cold temperatures	`load_temperatures()` + `categorize_temperatures()`	2
Total			10

□ Visible Test Case Examples

□ TC1 – Load Temperatures
Input:

```
df = converter.load_temperatures([10, 20, 30])
df.show()
```

Output:

```
diff

+-------+
|celsius|
+-------+
|   10.0|
|   20.0|
|   30.0|
+-------+
```

□ TC2 – Convert to Fahrenheit
Input:

```
df = converter.convert_to_fahrenheit(df)
df.show()
```

Output:

```
diff

+-------+----------+
|celsius|fahrenheit|
+-------+----------+
|   10.0|      50.0|
|   20.0|      68.0|
|   30.0|      86.0|
+-------+----------+
```

□ TC3 – Average
Input:

```
converter.calculate_average(df)
```

Output:

20.0