✅ Step-by-Step: Enum Testing with `@EnumSource` (Simplified Version)

📁 1. Project Structure

```
src/
├── main/
│   └── java/
│       └── com/
│           └── example/
│               └── Day.java          ✅ Enum class (simplified)
├── test/
│   └── java/
│       └── com/
│           └── example/
│               └── DayTest.java      ✅ JUnit 5 test class
```

📄 2. `Day.java` — Simplified Enum

```
package com.example;
public enum Day {
    MONDAY,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,
    SATURDAY,
    SUNDAY
}
```

This enum has no extra logic, just the names — perfect for basic enum source testing.

📄 3. `DayTest.java` — JUnit 5 Test Class

```
package com.example;

import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.EnumSource;
import static org.junit.jupiter.api.Assertions.*;
```

```java
public class DayTest {

    // Test 1: All enum constants

    @ParameterizedTest
    @EnumSource(Day.class)
    void testAllDaysNotNull(Day day) {
        assertNotNull(day);
    }

    // Test 2: Filter specific days

    @ParameterizedTest
    @EnumSource(value = Day.class, names = {"MONDAY", "FRIDAY"})
    void testSpecificDays(Day day) {
        assertTrue(day == Day.MONDAY || day == Day.FRIDAY);
    }

    // Test 3: Exclude weekends

    @ParameterizedTest
    @EnumSource(value = Day.class, names = {"SATURDAY", "SUNDAY"}, mode =
EnumSource.Mode.EXCLUDE)
    void testWeekdays(Day day) {
        assertFalse(day == Day.SATURDAY || day == Day.SUNDAY);
    }

    // Test 4: Match names with regex

    @ParameterizedTest
    @EnumSource(value = Day.class, names = ".*DAY", mode =
EnumSource.Mode.MATCH_ALL)
    void testRegexPattern(Day day) {
        assertTrue(day.name().endsWith("DAY"));
    }
}
```

🛠️ Gradle: `build.gradle`

```groovy
dependencies {
    testImplementation 'org.junit.jupiter:junit-jupiter-api:5.10.0'
    testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine:5.10.0'
}

test {
    useJUnitPlatform()
}
```

✅ Summary

File	Purpose
`Day.java`	Enum class with 7 constants
`DayTest.java`	JUnit test using `@EnumSource`
`pom.xml` or `build.gradle`	Dependency management

Would you like a zipped version of this whole project or should I walk you through creating and running it in IntelliJ or Eclipse?

✅ `@EnumSource` Syntax Variants

`@EnumSource` is used with `@ParameterizedTest` to provide enum constants as parameters.

🔹 1. Basic Usage – All Enum Constants

Syntax:

```
@EnumSource(MyEnum.class)
```

Example:

```
@ParameterizedTest
@EnumSource(Day.class)
void testAllDays(Day day) {
    assertNotNull(day);
}
```

✅ Runs the test once for each constant in `Day`.

🔹 2. Specify Enum Values by Name

Syntax:

```
@EnumSource(value = MyEnum.class, names = {"ENUM1", "ENUM2"})
```

Example:

```
@ParameterizedTest
@EnumSource(value = Day.class, names = {"MONDAY", "FRIDAY"})
void testSpecificDays(Day day) {
    assertTrue(day == Day.MONDAY || day == Day.FRIDAY);
}
```

✅ Runs test only for the specified enum constants.

🔹 3. Exclude Specific Enum Values

Syntax:

```
@EnumSource(value = MyEnum.class, names = {"ENUM1"}, mode =
EnumSource.Mode.EXCLUDE)
```

Example:

```
@ParameterizedTest
@EnumSource(value = Day.class, names = {"SUNDAY"}, mode =
EnumSource.Mode.EXCLUDE)
void testWeekdays(Day day) {
    assertNotEquals(Day.SUNDAY, day);
}
```

✅ Excludes the specified enum values from the test run.

🔹 4. Match Enum Values Using Regex

Syntax:

```
@EnumSource(value = MyEnum.class, names = ".*PART", mode =
EnumSource.Mode.MATCH_ALL)
```

Example:

```
@ParameterizedTest
```

```
@EnumSource(value = Day.class, names = ".*DAY", mode =
EnumSource.Mode.MATCH_ALL)
void testRegexPattern(Day day) {
    assertTrue(day.name().endsWith("DAY"));
}
```

✅ Uses a regex pattern to match enum names.

✅ EnumSource.Mode Values

Mode	Description
`INCLUDE` (default)	Use **only** the listed enum names.
`EXCLUDE`	Use **everything except** the listed names.
`MATCH_ALL`	Use **regex** to match multiple constants.

✅ Summary Table

Use Case	Syntax Example
All enum values	`@EnumSource(MyEnum.class)`
Only specific values	`@EnumSource(value = MyEnum.class, names = {"ONE", "TWO"})`
Exclude specific values	`@EnumSource(value = MyEnum.class, names = {"TWO"}, mode = EnumSource.Mode.EXCLUDE)`
Regex match	`@EnumSource(value = MyEnum.class, names = ".*DAY", mode = EnumSource.Mode.MATCH_ALL)`

Would you like me to generate a class that shows all of these syntaxes together in one place?

✅ Code:

```
@ParameterizedTest
@EnumSource(value = Day.class, names = ".*DAY", mode =
EnumSource.Mode.MATCH_ALL)
void testRegexPattern(Day day) {
    assertTrue(day.name().endsWith("DAY"));
}
```

🔍 What is it doing?

- `@ParameterizedTest`: This tells JUnit to run the test **multiple times**, each time with a different value.
- `@EnumSource(...)`: Specifies that the input values come from the `Day` enum.
- `value = Day.class`: Indicates the enum class `Day` is the source.
- `names = ".*DAY"`: This is a **regular expression** that matches any enum constant **whose name ends with `DAY`**.
- `mode = EnumSource.Mode.MATCH_ALL`: Tells JUnit to interpret the `names` value as a **regex pattern** (not literal names).

📄 Example `Day` Enum:

```
enum Day {
    MONDAY,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,
    SATURDAY,
    SUNDAY
}
```

✅ What values will be passed to the test?

Every enum constant in `Day` ends with `"DAY"`, so the regex `".*DAY"` matches **all** of them:

- `MONDAY`
- `TUESDAY`
- `WEDNESDAY`
- `THURSDAY`

- `FRIDAY`
- `SATURDAY`
- `SUNDAY`

✅ What will the test do?

For each of these enum constants, it checks:

```
assertTrue(day.name().endsWith("DAY"));
```

Which will always pass, because every enum constant in `Day` **does** end with `"DAY"`.

✅ Output (Conceptually):

The test runs 7 times (once per day):

```
✔ testRegexPattern(Day.MONDAY)
✔ testRegexPattern(Day.TUESDAY)
✔ testRegexPattern(Day.WEDNESDAY)
✔ testRegexPattern(Day.THURSDAY)
✔ testRegexPattern(Day.FRIDAY)
✔ testRegexPattern(Day.SATURDAY)
✔ testRegexPattern(Day.SUNDAY)
```

All tests **pass** ✅.

Would you like a failing example too, where the enum doesn't match the pattern?

@EnumSource(value = Day.class, names = ".*DAY", mode = EnumSource.Mode.MATCH_ALL)

is telling JUnit:

“Run this test for all Day enum constants whose names match the regular expression .*DAY (i.e., names that end with 'DAY').”

So yes — it selects and tests all enums ending with DAY.
In your case, since all enum constants in Day end with "DAY", the test runs for all of them.

If your enum had other constants like:

```
enum Day {
    MONDAY, TUESDAY, SPECIAL, CUSTOM
}
```

Then only MONDAY and TUESDAY would be selected for testing with that @EnumSource.

Would you like to see how the test behaves with non-matching enums too?