Enums

Enums (short for enumerations) are a powerful yet underappreciated feature in object-oriented design. They allow you to define a fixed set of named constants that improve clarity, type safety, and maintainability in your system.

Used correctly, enums can make your code more expressive, self-documenting, and resilient to errors.

What is an Enum?

An enum is a special data type that defines a collection of constant values under a single name. Unlike primitive constants or string literals, enums are type-safe; you can't assign just any value to a variable declared as an enum type.

Enums are perfect when a variable can only take one out of a small set of predefined values.

Example Enums

States (e.g., PENDING, IN_PROGRESS, COMPLETED)
Roles (e.g., ADMIN, CUSTOMER, DRIVER)
Vehicle Types (e.g., CAR, BIKE, TRUCK)
Directions (e.g., NORTH, SOUTH, EAST, WEST)

Enums help avoid magic strings or numbers, improve readability, enable compiler checks, IDE auto-completion and reduce bugs caused by invalid values.

Examples

Simple Enum
Enum representing status of an order in an e-commerce application.

Java

```
public enum OrderStatus {
    PLACED,
    CONFIRMED,
    SHIPPED,
    DELIVERED,
    CANCELLED
}
```

Python

```
from enum import Enum

# Simple Enum
class OrderStatus(Enum):
    PLACED = "PLACED"
    CONFIRMED = "CONFIRMED"
    SHIPPED = "SHIPPED"
    DELIVERED = "DELIVERED"
    CANCELLED = "CANCELLED"
```

This defines a clear set of valid statuses for an order.

Using it in code:

Java

```
OrderStatus status = OrderStatus.SHIPPED;

if (status == OrderStatus.SHIPPED) {
    System.out.println("Your package is on the way!");
}
```

Python

```
status = OrderStatus.SHIPPED

if status == OrderStatus.SHIPPED:
    print("Your package is on the way!")
```

Enums with Properties and Methods
Enums can have additional data and even behavior. This makes them even more powerful.

Coin Enum with Denomination

Java

```
public enum Coin {
    PENNY(1),
```

```
    NICKEL(5),
    DIME(10),
    QUARTER(25);

    private final int value;

    Coin(int value) {
        this.value = value;
    }

    public int getValue() {
        return value;
    }
}
```

Python

```
from enum import Enum

class Coin(Enum):
    PENNY = 1
    NICKEL = 5
    DIME = 10
    QUARTER = 25

    def __init__(self, value):
        self.coin_value = value

    def get_value(self):
        return self.coin_value
```

Usage:

Java

```
int total = Coin.DIME.getValue() + Coin.QUARTER.getValue(); // 35
```

Python

```
total = Coin.DIME.get_value() + Coin.QUARTER.get_value()  # 35
```

This is far more elegant and safe than using integers directly.