10.10-Enum Class

In Java, an **enum** is essentially a class, but it cannot be extended like a regular class. Despite this, the enum class possesses many of the properties that a normal class has. For example, we can:

- Create methods
- Define constructors
- Declare and define variables

Let's explore enums further, focusing on inheritance, constructors, and methods.

Enum and Inheritance

Although enums in Java cannot extend any other class, they implicitly extend the java.lang.Enum class. This is why enums cannot extend any other class (since Java doesn't support multiple inheritance with classes).

Key points regarding inheritance with enums:

- Enums inherit from the Enum class, which is a direct subclass of the Object class.
- The toString() method is overridden in the Enum class, returning the name of the enum constant.
- Enums **can** implement interfaces, allowing enums to behave like normal classes in that respect.

Enum and Constructors

Enums can have constructors, which are called separately for each enum constant at the time the enum is loaded. There are some specific behaviors regarding constructors in enums:

- **Private Constructors:** Enum constructors are always private. Even though you can define constructors, you cannot instantiate enums directly using the new keyword. The enum constants are the only instances of the enum type.

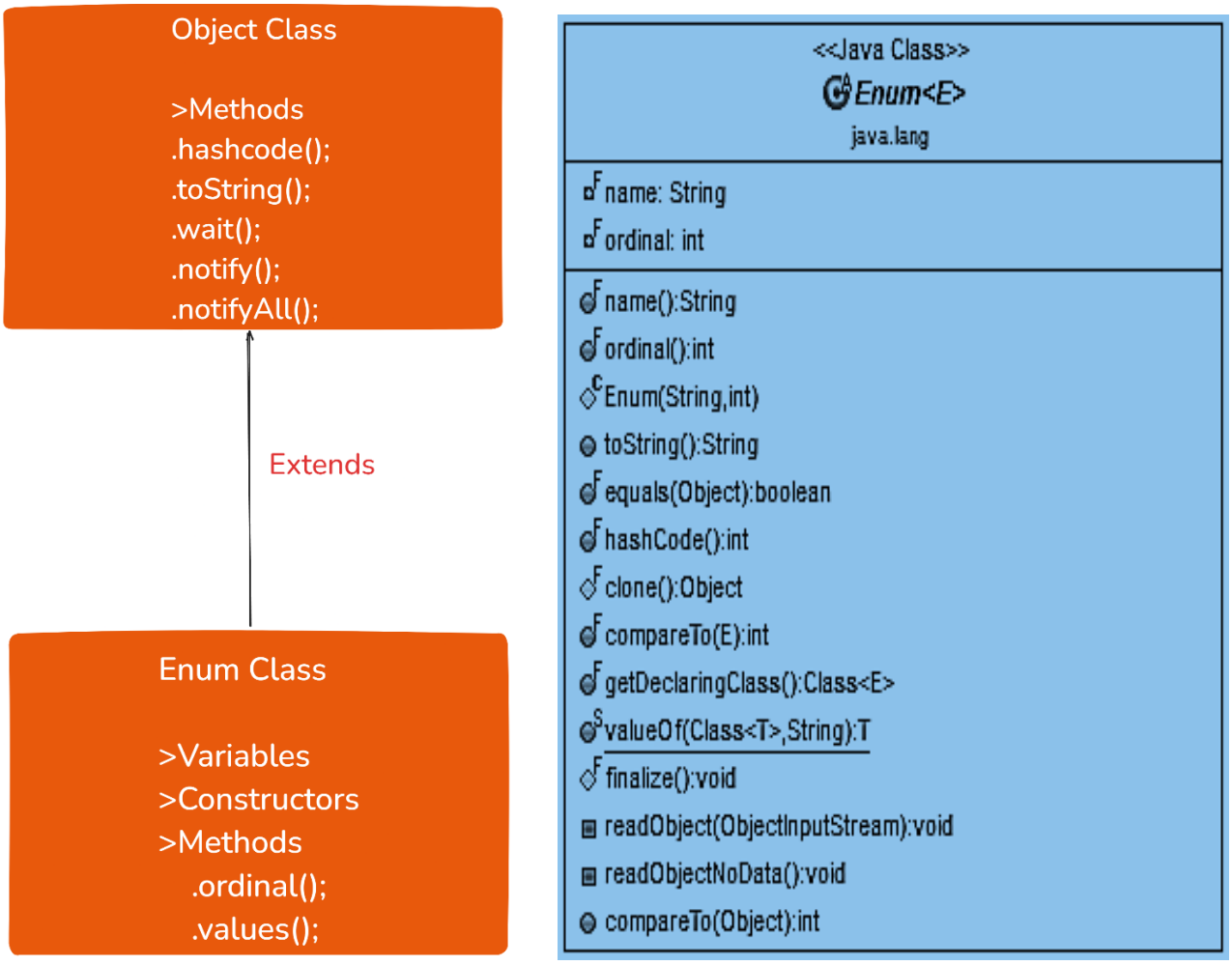
- **Parameterized Constructors:** You can use constructors to assign values to enum constants. For example, you might want to assign a price to a laptop brand in an enum.

Important: If you define a parameterized constructor, all enum constants must be initialized with corresponding values. If you omit values, you will get a compilation error.

Methods in Enum

Enums have several built-in methods, many of which come from the Enum class. Some common methods include:

- **ordinal()** – Returns the position of the enum constant.
- **values()** – Returns an array of all enum constants.



Enums can also have user-defined methods, both concrete and abstract. If you include abstract methods in an enum, each enum constant must provide an implementation for them.

Example of Enum with Constructors and Methods

Let's look at an example to illustrate how enums work, including how to use constructors and methods in enums.

```
enum Laptop {
    Macbook(2000), XPS(2200), Surface(2200),
ThinkPad(1500);
    private int price;

    // Parameterized constructor
    private Laptop(int price) {
        this.price = price;
        System.out.println("Laptop: " + this.name());
    }

    // Getter for price
    public int getPrice() {
        return price;
    }

    // Setter for price
    public void setPrice(int price) {
        this.price = price;
    }
}

public class Demo {
    public static void main(String[] args) {
        Laptop lap = Laptop.Macbook;
        System.out.println("Selected Laptop: " + lap);

        // Display all enum constants with their prices
        for (Laptop l : Laptop.values()) {
            System.out.println(l + ": $" + l.getPrice());
        }
    }
}
```

Output:

```
Laptop: Macbook
Selected Laptop: Macbook
Macbook: $2000
XPS: $2200
Surface: $2200
ThinkPad: $1500
```

Explanation of Example

1. **Enum Constants with Prices:** In the Laptop enum, we define four constants: Macbook, XPS, Surface, and ThinkPad. Each of these constants is initialized with a price using the parameterized constructor.
2. **Private Variables and Constructor:** The enum has a private variable price, which stores the price for each constant. The constructor takes an int argument to initialize the price for each laptop.
3. **Getter and Setter Methods:** Since the price is private, we use a getter (getPrice()) to access the price value. We also provide a setter (setPrice()) to allow the modification of prices if necessary.
4. **Looping Through Enum Constants:** The values() method is used to loop through all the enum constants (Laptop.values()) and print each constant's name and its price.

Common Issues and Solutions

- **Omitting Values:** If you define a parameterized constructor for your enum but forget to initialize one of the constants with a value, the compiler will throw an error. To avoid this, either assign a value to every constant or provide a default constructor.
- **Constructor Visibility:** The constructor in an enum is always private. You do not need to explicitly specify the private keyword, but it is good practice to do so to make the visibility clear.