
🍕 The Backend Pizzeria Quest – The Complete Adventure (100 + 10 Bonus pts)

A Java 21 console-based saga in SOLID and design patterns.

🌟 PROLOGUE – *The Curse of the Spaghetti Sorcerer*

In the kingdom of **Codonia**, one class — `PizzaApp` — controls all: oven, prices, printing, and conversation.
King **SOLIDius** summons you, **Patrizio the Refactorer**, to rebuild the royal pizzeria into a clean, modular masterpiece.

🧙‍♂️ LEVEL 1 – *The Swamp of Magic Literals* (15 pts)

Quest
Replace every magic number and string with clear, named constants.

Tasks

1. Create `PizzaConfig` (class / interface) containing:
    - Pizza names & base prices
    - Tax rate
    - Currency symbol
2. Update all other code to use these constants.
3. JUnit test ensures `TAX_RATE == 0.19`.

Reward: 15 pts
Commit: `Level 1 – literals banished`

🏗️ LEVEL 2 – *The Collosus-Class Giant Falls* (20 pts)

Quest
Apply **Single Responsibility** and build a layered architecture.

Packages

rotten.pizza
├── app (Main – console menu)
├── model (Pizza & ingredients)
├── service (business logic)
└── util (optional helpers)

Key roles

- `model.Pizza` → interface
- concrete classes → `Margherita`, `Pepperoni`, `Hawaiian`
- `service.PriceCalculator` → handles prices & tax
- `app.Main` → console flow (≤ 20 lines)

Reward: 20 pts
Commit: `Level 2 - giant split into craftsmen`

🍕 PIZZA MODEL DESIGN GUIDE

```java
// model/Pizza.java
package rotten.pizza.model;

public interface Pizza {
    String getName();
    double getBasePrice();
}
```

Example implementation:

```java
// model/Margherita.java
package rotten.pizza.model;

public class Margherita implements Pizza {
    @Override public String getName() { return "Margherita"; }
    @Override public double getBasePrice() { return PizzaConfig.MARGHERITA_PRICE; }
}
```

✅ Use **classes**, not enums — allows later extensions (ingredients, calories, etc.).

🔮 LEVEL 3 – *The Ever-Changing Menu* (15 pts)

Quest
Use **Factory Method** so new pizzas can be added without touching existing code.

Tasks

1. Interface `PizzaFactory` with `Pizza create(String name)`.
2. Class `MenuPizzaFactory` using a `Map<String, Supplier<Pizza>>`.
3. Adding a pizza = new class + one registration line.
4. Add and test `"QuattroFormaggi"`.

Reward: 15 pts
Commit: `Level 3 – menu open for extension`

💰 LEVEL 4 – *The Strategy of Discounts* (25 pts)

Quest
Introduce **Strategy Pattern** for flexible discount rules.

Tasks

1. Interface `PricingStrategy` → `double apply(double net)`.
2. Implement:
    - `RegularPricing` (no discount)
    - `StudentPricing` (–10 %)
    - `HappyHourPricing` (–20 %)
3. Inject strategy into `PriceCalculator`.
4. **Instead of CLI flags**, create a **Console Adventure Menu**:
    - Ask user which discount applies.

Example:

Choose your deal:
1) Regular
2) Student (-10%)
3) Happy Hour (-20%)

    - Then apply the chosen strategy.

5. Unit-test each strategy class.

Reward: 25 pts
Commit: `Level 4 – pricing strategies interchangeable`

🧱 LEVEL 5 – *The Toppings Guild (Composite Pattern)* (25 pts)

Story
The pizzeria now serves combo pizzas and half-and-half specials.
You must let multiple pizzas behave like one.

Quest
Use the **Composite Pattern**.

Tasks

1. Extend `Pizza` to support both single and composed items.

Create:

```
public class CompositePizza implements Pizza {
    private final List<Pizza> components = new ArrayList<>();
    public void add(Pizza p) { components.add(p); }
    @Override public String getName() {
        return components.stream()
            .map(Pizza::getName)
            .collect(Collectors.joining(" + "));
    }
    @Override public double getBasePrice() {
        return components.stream()
            .mapToDouble(Pizza::getBasePrice)
            .sum();
    }
}
```

2.
3. Demonstrate a “Half & Half” pizza and verify total pricing.

Reward: 25 pts
Commit: `Level 5 – composite pizzas created`

✨ BONUS LEVEL 6 – *The Receipt Enchanter (Decorator Pattern)* (+10 pts)

Story
The Queen wants beautifully formatted receipts — borders, emojis, coupons — but changing the printer each time breaks the Open/Closed rule.

Quest
Use the **Decorator Pattern** to format receipts dynamically.

Tasks

1. Create `ReceiptPrinter` interface with `String print(Pizza pizza, double price)`. Example:

2. Implement:

    - `BasicReceiptPrinter` → plain text
    - `EmojiReceiptDecorator` → adds emoji borders
    - (optional) `DiscountBannerDecorator` → adds banner lines

In `Main`, wrap printers:

```
ReceiptPrinter printer =
    new EmojiReceiptDecorator(new BasicReceiptPrinter());
System.out.println(printer.print(pizza, finalPrice));
```

3. Show that different decorators change output without editing old classes.
4. Combining Decorators

```
ReceiptPrinter printer =
    new CouponDecorator(
        new EmojiReceiptDecorator(
            new BasicReceiptPrinter()
        )
    );

Pizza pizza = new Margherita();
double finalPrice = 12.50;
System.out.println(printer.print(pizza, finalPrice));
```

Prints:

Reward: +10 Bonus pts
Commit: `Level 6 – receipt decorator enchanted`

🏆 EPILOGUE – *The SOLID Kingdom Restored*

Principle	What You Did
S	One reason to change per class
O	Extend code without modifying existing files
L	Interchangeable objects (`Pizza`, `PricingStrategy`)
I	Focused interfaces
D	Depend on abstractions, not details

🗒️ Scoring Summary

Level	Title	Pattern / Concept	Points
1	Swamp of Magic Literals	Constants & Readability	15
2	Collosus-Class Giant	SRP + Layered Structure	20
3	Ever-Changing Menu	Factory Method	15
4	Strategy of Discounts	Strategy Pattern (+ Console Menu)	25
5	Toppings Guild	Composite Pattern	25
Total			**100 pts**
Bonus 6	Receipt Enchanter	Decorator Pattern	**+10 pts**
