Decorator Design Pattern

What is Decorator Design Pattern?

The *Decorator Design Pattern* is a structural pattern that provides a ***flexible alternative*** to ***subclassing*** for **extending functionality**. It ***avoids class explosion*** by allowing you to **dynamically *add responsibilities*** to objects at *runtime* **without modifying existing code** or **creating many subclasses**.

Let's explore how the **Decorator Design Pattern** solves the **inheritance explosion problem** in a ***Mario game***, where Mario gains new abilities like:

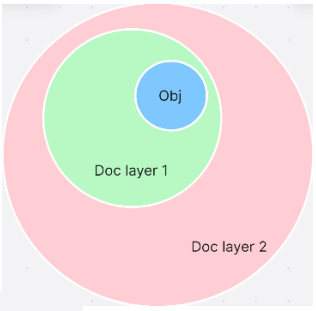
- 🍄 Height increase (Super Mario)
- 🔫 Gun power (Fire Mario)
- ⭐ Time-limited invincibility (Star Mario)

If we used inheritance to represent each **ability combination**, we would get something like:

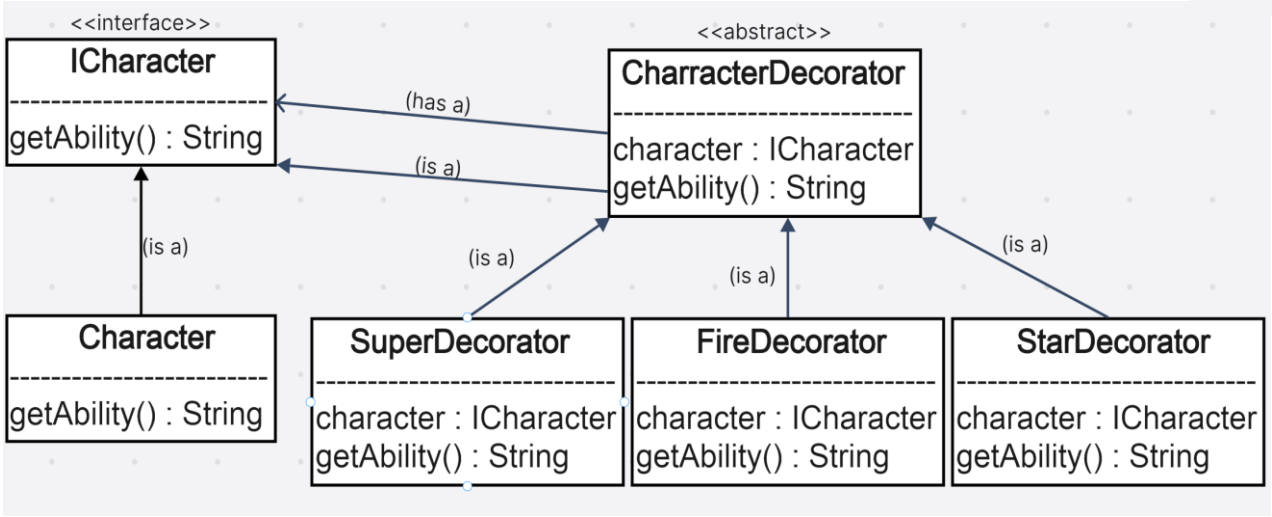
→ Mario
→ **Super**Mario
→ **Fire**Mario
→ **Star**Mario
→ **Super*Fire***Mario
→ **Super*Star***Mario
→ **Fire*Star***Mario
→ **Super*Fire*Star**Mario
...and so on.

❌ This leads to *class explosion* because every possible combination needs a **new subclass**.

✅ Instead of **creating** a **new subclass** for every combination, we start with a **base class** **Mario**, and dynamically **wrap decorators** around it to **add powers**.



Sample UML diagram: correct and clean implementation of the Decorator Design Pattern for the Mario example.



Code Link:- https://github.com/sibasundarj8/-System-Design-/tree/main/Codes/13___Decorator%20Design%20Pattern%20code

@Sibasundarj8