The **decorator design pattern** allows dynamically adding functionality to an existing object without altering its original class. [It achieves this by wrapping the original object in a decorator class that has the same interface as the object it decorates1](https://www.ionos.com/digitalguide/websites/web-development/what-is-the-decorator-pattern/)[2](https://www.pentalog.com/blog/design-patterns/decorator-design-pattern/).

Here are **five free reference links** where you can learn more about the decorator design pattern:

1. **IONOS**: Provides an explanation, UML presentation, and example of the decorator pattern: [Decorator pattern: explanation, UML presentation, and example](https://www.ionos.com/digitalguide/websites/web-development/what-is-the-decorator-pattern/)
2. **GeeksforGeeks**: Offers a Java tutorial with examples on implementing the decorator pattern: [Decorator Method Design Patterns in Java with Example](https://www.geeksforgeeks.org/decorator-design-pattern-in-java-with-example/)
3. **Visual Paradigm**: A tutorial with step-by-step guidance on applying the decorator pattern: [Decorator Pattern Tutorial](https://www.visual-paradigm.com/tutorials/decoratordesignpattern.jsp)
4. **Scaler Topics**: Learn design patterns, including the decorator pattern, through Scaler’s comprehensive guide: [Design Patterns Tutorial](https://www.scaler.com/topics/design-patterns/)
5. **SO Documentation**: A concise explanation of the decorator pattern: [Design patterns Tutorial => Decorator pattern](https://riptutorial.com/design-patterns/topic/1720/decorator-pattern)

Feel free to explore these resources to deepen your understanding of this powerful design pattern! 🚀