perplexity

Difference between all the Structural design pattern

Here is a concise comparison of the **7 Gang of Four Structural Design Patterns**, explaining their **purpose, key idea, and differences** based on the search results:

Pattern	Purpose	Key Idea / How It Works	When to Use / Where
Adapter	Allows two incompatible interfaces to work together by wrapping one interface around the other.	Acts as a bridge converting one interface into another expected by the client.	When integrating legacy or third-party code with incompatible interfaces.
Bridge	Decouples an abstraction from its implementation so both can vary independently.	Splits a class into two hierarchies: abstraction and implementation, connected via composition.	When both abstraction and implementation may change independently; to avoid class explosion.
Composite	Composes objects into tree structures to represent part-whole hierarchies.	Treats individual objects and compositions uniformly via a common interface.	When you want to represent hierarchical data and treat leaves and composites uniformly.
Decorator	Attaches additional responsibilities to an object dynamically without subclassing.	Wraps an object in decorator objects that add new behavior, all sharing the same interface.	When you want to add functionality dynamically and flexibly without subclass explosion.
Facade	Provides a simplified interface to a complex subsystem.	Offers a high-level interface that hides subsystem complexity from the client.	When you want to hide complex subsystem details and provide a simple entry point.
Flyweight	Minimizes memory usage by sharing common parts of many fine-grained objects.	Shares intrinsic state among many objects, storing extrinsic state externally.	When you have large numbers of similar objects and want to reduce memory footprint.
Proxy	Provides a surrogate or placeholder to control access to another object.	Implements the same interface as the real object and controls access, adding behavior like caching or security.	When you want to add access control, lazy loading, logging, or remote access without changing the real object.

Key Differences and Relationships

- **Adapter vs Bridge:**
  Adapter converts one interface to another and is applied when integrating existing incompatible classes; Bridge separates abstraction and implementation to allow independent evolution.

- **Composite vs Decorator:**
  Both use recursive composition, but Composite builds tree structures for part-whole hierarchies, while Decorator builds chains to add responsibilities dynamically.
- **Proxy vs Decorator:**
  Both delegate to a real object, but Proxy controls access (security, lazy loading), whereas Decorator adds new behavior or responsibilities.
- **Facade vs Others:**
  Facade simplifies a complex subsystem by providing a single unified interface, whereas others focus on object composition and behavior.

Summary

Pattern	Focus	Main Benefit	Example Use Case
Adapter	Interface compatibility	Reuse incompatible classes	Legacy API integration
Bridge	Decouple abstraction & impl	Independent extensibility	GUI toolkit with multiple rendering engines
Composite	Part-whole hierarchy	Uniform treatment of individual and composite	File system directories and files
Decorator	Dynamic behavior extension	Flexible addition of responsibilities	Adding scrollbars or borders to UI elements
Facade	Simplified interface	Hide subsystem complexity	Simplified database API
Flyweight	Memory optimization	Share common state among many objects	Text editor characters
Proxy	Controlled access	Add access control, caching, or lazy loading	Virtual proxy for loading images

This overview synthesizes authoritative explanations from DigitalOcean, Refactoring.Guru, GeeksforGeeks, and academic comparisons.

If you want, I can provide detailed example codes or deeper explanations for any specific structural pattern.

⁂