**🏗️ 1. Creational Patterns**

**Mnemonic**: **"Silly Architects Build Fancy Prototypes"**

| **Pattern** | **Purpose** | **Java Tip** | **Analogy** |
| --- | --- | --- | --- |
| **Singleton** | One instance only | private static + getInstance() | CEO of a company |
| **Abstract Factory** | Create families of related objects | Factory of factories | Car factory producing engines & tires |
| **Builder** | Step-by-step object creation | Fluent API | Making a pizza with toppings |
| **Factory Method** | Delegate instantiation to subclasses | createProduct() | Hiring via HR department |
| **Prototype** | Clone existing objects | clone() method | Copying a document |

**🧱 2. Structural Patterns**

**Mnemonic**: **"A Big Fat CD Plays With Style"**

| **Pattern** | **Purpose** | **Java Tip** | **Analogy** |
| --- | --- | --- | --- |
| **Adapter** | Convert one interface to another | Wrapper class | Power plug converter |
| **Bridge** | Separate abstraction from implementation | Interface + Implementation | Remote control for different TVs |
| **Facade** | Simplify complex subsystem | One entry-point class | Hotel reception desk |
| **Composite** | Treat group and individual uniformly | Tree structure | Folder with files |
| **Decorator** | Add behavior dynamically | Wrap original object | Gift wrapping |
| **Proxy** | Control access to object | Virtual or remote proxy | Security guard |
| **Flyweight** | Share common state | Cache shared objects | Chess pieces reused |

**🤝 3. Behavioral Patterns**

**Mnemonic**: **"Some Clever Coders Often Make Software That Is Very Clean & Manageable"**

| **Pattern** | **Purpose** | **Java Tip** | **Analogy** |
| --- | --- | --- | --- |
| **Strategy** | Swap algorithms at runtime | Interface + multiple implementations | Payment method selection |
| **Command** | Encapsulate a request | execute() method | Remote control buttons |
| **Observer** | Notify dependent objects | addObserver() | News subscription |
| **Mediator** | Centralize communication | Chatroom class | Air traffic controller |
| **State** | Change behavior based on state | State interface | Traffic light |
| **Template Method** | Define algorithm skeleton | Abstract class | Cooking recipe steps |
| **Iterator** | Traverse collection | Iterator<T> | TV channel surfing |
| **Visitor** | Separate algorithm from object | accept(visitor) | Tax auditor visiting departments |
| **Chain of Responsibility** | Pass request along handlers | Linked handlers | Customer support escalation |
| **Interpreter** | Grammar interpreter | Expression parser | Calculator input |
| **Memento** | Save and restore state | Caretaker + Memento | Undo in text editor |

**🧠 Tips to Remember**

* **Group by purpose**: Creation, Structure, Behavior.
* Use **real-world analogies** to anchor memory.
* Practice with **Java examples** in small projects.
* Apply patterns in **enterprise scenarios** like billing, workflow engines, or device management.