Design patterns in Java are best practices that provide solutions to common software design problems. They are categorized into three main types: Creational, Structural, and Behavioral patterns. Here’s a brief overview of each:

**1. Creational Design Patterns**

These patterns deal with object creation mechanisms, trying to create objects in a manner suitable to the situation.

* **Singleton Pattern**: Ensures a class has only one instance and provides a global point of access to it.
* **Factory Method Pattern**: Defines an interface for creating an object, but lets subclasses alter the type of objects that will be created.
* **Abstract Factory Pattern**: Provides an interface for creating families of related or dependent objects without specifying their concrete classes.
* **Builder Pattern**: Separates the construction of a complex object from its representation, allowing the same construction process to create different representations.
* **Prototype Pattern**: Creates new objects by copying an existing object, known as the prototype.

**2. Structural Design Patterns**

These patterns deal with object composition, ensuring that if one part changes, the entire structure does not need to change.

* **Adapter Pattern**: Allows incompatible interfaces to work together.
* **Composite Pattern**: Composes objects into tree structures to represent part-whole hierarchies.
* **Proxy Pattern**: Provides a surrogate or placeholder for another object to control access to it.
* **Decorator Pattern**: Adds additional responsibilities to an object dynamically.
* **Facade Pattern**: Provides a simplified interface to a complex subsystem.
* **Flyweight Pattern**: Reduces the cost of creating and manipulating a large number of similar objects.

**3. Behavioral Design Patterns**

These patterns are concerned with algorithms and the assignment of responsibilities between objects.

* **Observer Pattern**: Defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.
* **Strategy Pattern**: Defines a family of algorithms, encapsulates each one, and makes them interchangeable.
* **Command Pattern**: Encapsulates a request as an object, thereby allowing for parameterization of clients with queues, requests, and operations.
* **State Pattern**: Allows an object to alter its behavior when its internal state changes.
* **Template Method Pattern**: Defines the skeleton of an algorithm in a method, deferring some steps to subclasses.
* **Chain of Responsibility Pattern**: Passes a request along a chain of handlers, allowing each handler to process the request or pass it to the next handler in the chain.

These patterns help in making code more flexible, reusable, and easier to maintain [[1]](https://www.geeksforgeeks.org/java-design-patterns/) [[2]](https://www.digitalocean.com/community/tutorials/java-design-patterns-example-tutorial) [[3]](https://www.tutorialspoint.com/design_pattern/index.htm).

If you have any specific design pattern in mind or need examples, feel free to ask!

**References**

[1] [Java Design Patterns Tutorial - GeeksforGeeks](https://www.geeksforgeeks.org/java-design-patterns/)

[2] [Most Common Design Patterns in Java (with Examples)](https://www.digitalocean.com/community/tutorials/java-design-patterns-example-tutorial)

[3] [Design Patterns in Java Tutorial - Online Tutorials Library](https://www.tutorialspoint.com/design_pattern/index.htm)