

is-a → inheritance
has-a → composition

```

classDiagram
    class Client
    class Iterable {
        create_iterator()
    }
    class Iterator {
        has_next()
        next()
    }
    class ConcreteIterable {
        create_iterator()
    }
    class ConcreteIterator {
        has_next()
        next()
    }
    Client --> Iterable
    Client --> Iterator
    Iterable <|-- ConcreteIterable
    Iterator <|-- ConcreteIterator
    ConcreteIterable --> ConcreteIterator
  
```

The diagram illustrates the Iterator Pattern with the following components and relationships:

- Client**: Interacts with both the **Iterable** and **Iterator** interfaces.
- Iterable**: An interface with a `create_iterator()` method.
- Concrete Iterable**: Implements the **Iterable** interface.
- Iterator**: An interface with `has_next()` and `next()` methods.
- Concrete Iterator**: Implements the **Iterator** interface.

Handwritten notes at the bottom identify the concrete classes: "Hand Handing Iterable Backpack Inventory" for **Concrete Iterable** and "Hand Handing Iterator Backpack Iterator" for **Concrete Iterator**.

[illegible]

UML Diagram of the Decorator Pattern:

```

classDiagram
    class Component {
        method A()
        method B()
    }
    class ConcreteComponent {
        method A()
        method B()
    }
    class Decorator {
        method A()
        method B()
    }
    class ConcreteDecorator1 {
        method A()
        method B()
    }
    class ConcreteDecorator2 {
        method A()
        method B()
    }
    Component <|-- ConcreteComponent
    Component <|-- Decorator
    Decorator <|-- ConcreteDecorator1
    Decorator <|-- ConcreteDecorator2
    Decorator o-- Component
  
```

Handwritten notes on the right side of the diagram:

- Decorator (purple circle)
- ConcreteComponent (yellow circle)
- Component (yellow circle)
- I have
- I am add
- I am remove

```
classDiagram
    class Client {
        +IBehavior behavior
        +execute()
    }
    class IBehavior {
        +run()
    }
    class ConcreteBehaviorA {
        +run()
    }
    class ConcreteBehaviorB {
        +run()
    }
    Client --> IBehavior
    IBehavior <|-- ConcreteBehaviorA
    IBehavior <|-- ConcreteBehaviorB
```

```
classDiagram
    class Animal {
        Product
    }
    class ConcreteProduct {
    }
    class AnimalFactory {
        createor
        factory method()
        an Operation()
    }
    class ConcreteFactory {
        createor
    }
    Animal <|-- ConcreteProduct
    AnimalFactory <|-- ConcreteFactory
    ConcreteFactory --> ConcreteProduct
```

```

graph TD
    subgraph Gate
        Context[Context]
        Context -- "I-I" --> IState[I State]
    end
    subgraph IStateBox [I State]
        IHandle[handle()]
        IHandle --> Context
    end
    subgraph ContextStateA [Context State A]
        CHandle[handle()]
        CHandle --> Context
        CHandle --> IStateBox
    end
    ContextStateA -- "openGateState" --> Context
    ContextStateA -.->|CloseGateState| ContextStateA
    Processing[Processing Gate State]

```

Singleton
static singleton instance
static getInstance()
Single.getInstance()

```

classDiagram
    class Invoker {
        setCommand(ICommand)
    }
    class ICommand {
        execute()
        unexecute()
    }
    class Command {
        execute()
        unexecute()
    }
    class Receiver {
        can be anything
    }
    Invoker --> "1" * "ICommand" : setCommand(ICommand)
    ICommand <|-- Command
    Receiver --> Command
  
```

The diagram shows a table with four columns and four rows. The top row is labeled 'Table'. Below the table, four arrows point to the four columns, indicating that each column represents a different data type or structure.

```
classDiagram
    class Client
    class ITarget {
        request()
    }
    class Adapter {
        request()
    }
    class Adaptee {
        specific request()
    }
    Client --> ITarget
    Adapter --|> ITarget
    Adapter --> Adaptee
```

UML class diagram illustrating the Adapter pattern:

- Client**: Interacts with **ITarget**.
- ITarget**: Interface defining the `request()` method.
- Adapter**: Implements **ITarget** and delegates the `request()` call to **Adaptee**.
- Adaptee**: Implements the `specific request()` method.

Handwritten notes in pink:

- Below **Adapter**: Adapter Bus, Adapter Car, Adapter Bike
- Next to **Adaptee**: specific request()