Structural Design Patterns

1 Adapter Design Pattern.

Convert the interface of a class into another interface that a client expects.

In simple terms:
- make incompatible interfaces work together.
- Act as a bridge between old and new code.

Adapter allows two incompatible interfaces to work together.

Why Adapter Exists.

:- you want to use a third-party library.
- you have a legacy code.
- Interfaces don't match.

Adapter solves this without modifying existing code.

Core Idea:-

Instead of changing client or existing class,
Create an Adapter, it translates calls from client → existing class.

Eg. wall socket → different shape.
Charger → different shape.
Adapter makes them compatible.
Neither socke nor ~~charger~~ chargers changes.

Without adapter:-

```
Class OldPaymentService{
  Void makePayment(int amount){
}
client expects
interface PaymentGateway{
  void pay(double amount);
}
```

- Interfaces don't match.
- you cannot change old PaymentService

Adapter Solution:

Target interface (Expected by client):-

```
interface PaymentGateway {
    void pay(double amount);
}
```

Adaptee (Existing / Legacy class):-

```
class OldPaymentService {
    void makePayment(int amount) {
        System.out.println("Paid " + amount);
    }
}
```

Adapter class:-

```
class PaymentAdapter implements PaymentGateway {
    private OldPaymentService oldService;
    PaymentAdapter(OldPaymentService oldService) {
        this.oldService = oldService;
    }
    public void pay(double amount) {
        oldService.makePayment((int) amount);
    }
}
```

Client Code:-

```
PaymentGateway gateway =
        new PaymentAdapter(new OldPaymentService());
gateway.pay(100.50);
```

Obj Adapter:

- uses composition.
- Adapter has-a adaptee.
- preferred approach

Class Adapter:

- uses inheritance.
- Limited by single inheritance (Java).