Behavioral - State Design Pattern (2. Örneğe mutlaka bak)

State tasarım kalıbının kullanılması, nesnelerin durumlarına bağlı değişen davranışlarının karmaşık koşul ve kontrol **(if/else, switch)** ifadeleriyle yönetilmesini önler. Bir nesnenin internal state'inde meydana gelecek değişimler sonrası runtime'da dynamic bir şekilde davranışlar göstermesi için kullanılır.Nesnenin internal state'ini taşıyan nesneye context diyeceğiz. Birden fazla davranış ve doğal olarak durum olabileceğinden, Context'in farklı durumlarına ulaşılabilmeli ve state'ler arasında ki Transitions'ları sağlayabilmesi gerekmektedir.

```
interface State {
   void doAction(Context context);
}
```

Context içerisinde State'i aggregate etmektedir;

```
class Context {
   private State state;

   public State getState() {
      return state;
   }

   public void setState(State state) {
      this.state = state;
   }
}
```

Concrete State'ler;

```
class ModifiedState implements State {
   @Override
   public void doAction(Context context) {
      System.out.println("State modified at the moment");
      context.setState(this);
   }
}
class DeletedState implements State{
   @Override
   public void doAction(Context context) {
      System.out.println("State deleted at the moment");
      context.setState(this);
   }
}
```

Main methodu;

```
public static void main(String[] args) {
   Context context = new Context();
   ModifiedState modifiedState = new ModifiedState();
   modifiedState.doAction(context);
   DeletedState deletedState = new DeletedState();
   deletedState.doAction(context);
}
```

Behavioral - State Design Pattern

```
interface PackageState{
   void next(Context pkg);
   void prev(Context pkg);
   void printStatus();
}
```

Context

```
class Context{
   private PackageState state = new OrderedState(); //henüz yeni sipariş edilmiş order
   public PackageState getState() {
      return state;
   }
   public void setState(PackageState state) {
      this.state = state;
   }
   void next(){
      state.next(this);
   }
   void prev(){
      state.prev(this);
   }
   void printStatus(){
      state.printStatus();
   }
}
```

Concrete Class'lar

```
class OrderedState implements PackageState{
    @Override
    public void next(Context pkg) {
        pkg.setState(new DeliveredState());
    }
    @Override
    public void prev(Context pkg) {
        System.out.println("The package is in its root state.");
    }
    @Override
    public void printStatus() {
        System.out.println("Package ordered, not delivered to the office yet.");
    }
}
class DeliveredState implements PackageState {
    @Override
    public void next(Context pkg) {
        pkg.setState(new ReceivedState());
    }
    @Override
    public void prev(Context pkg) {
        pkg.setState(new OrderedState());
    }
    @Override
    public void printStatus() {
        System.out.println("Package delivered to post office, not received yet.");
    }
}
class ReceivedState implements PackageState {
    @Override
    public void next(Context pkg) {
        System.out.println("This package is already received by a client.");
    }
    @Override
    public void prev(Context pkg) {
        pkg.setState(new DeliveredState());
    }
    @Override
    public void printStatus() {
        System.out.println("Package was received by client.");
    }
}
```

Main

```
public static void main(String[] args) {
    Context context = new Context();
    context.printStatus();
    context.next();
    context.printStatus();
    context.next();
    context.printStatus();
    context.next();
    context.printStatus();
}
```