Behavioral Patterns

1. 1.1. Memento Pattern

Memento Tasarım Deseni, geri alınabilir eylemlerin implement edilebilmesi için bir çözüm sunmaktadır.

- Bir nesnenin önceki durumlarını saklamak/depolamak amacıyla kullanılır.
- İmplementasyon detayını açığa çıkarmadan nesnenin durumunun kaydedilmesi ve yüklenmesine imkan tanır.

1.1. Senaryo

Günümüz dünyasında son kullanıcılar, her türlü arayüz ve uygulamada geri alma ya da bir önceki duruma dönme seçeneklerini talep etmektedir. Bir nesnenin gerektiğinde yedeğinin alınabilmesi, kullanıcı deneyimi açısından oldukça önemlidir.

Bir MMORPG oyunundaki karakterin sahip olduğu envanteri düşünelim.

- Oyuncu envanteri önemlidir. Belirli aralıklarla bir yedeği alınır, kopyalanır.
- Eğer oyun içerisinde bir **roll-back işlemi gerekirse,** tüm oyuncuların **belirli tarihteki kopya envanterleri** oyuna yüklenir.

1.2. Problem

- Durumu kaydedilecek nesneye `Originator` denmektedir.
- Nesnenin tüm kaydedilen durumları `CareTaker` içerisinde tutulur.
- İlgili nesnenin her bir kopyasına `Memento` adı verilir.
- `Memento` nesnesi, `CareTaker` sınıfına mümkün olduğunca az bilgi açığa çıkarmalıdır.
    - **Kapsülleme prensiplerinin** çiğnenmemesi adına, **Originator sınıfının yapısı** dışarıdan erişilir ve kopyalanabilir halde tutulmamalıdır.

1.3. Çözüm

- Durumu kopyalanacak `Originator` nesnesi, kendisini kopyalar.

- Kapsülleme prensipleri gözetilir. Nesnenin dışarıdan erişim suretiyle kopyalamasının önüne geçilir.

- Pattern, kopya nesne durumunun `Memento` adı verilen özel bir nesnede tutulmasını önerir.
- Üretilen tüm kopya nesneler (Memento nesneleri), Caretaker nesnesi içerisinde tutulur.

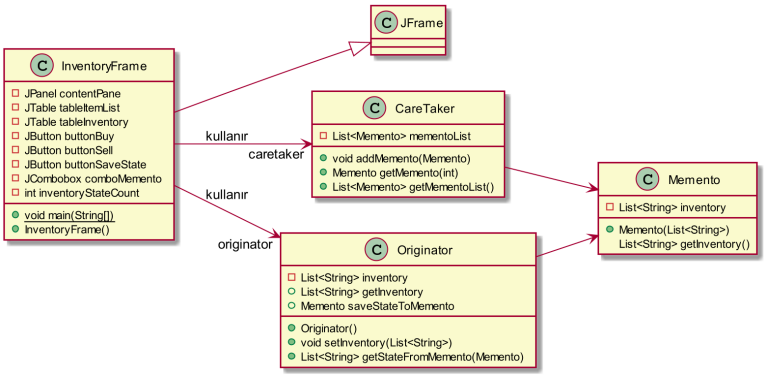


Figure 1. Memento tasarım kalıbı UML şeması

Originator.java.

```
public class Originator {
 private List<String> inventory = new ArrayList<>();

 public List<String> getInventory() {
  return inventory;
 }

 public void setInventory(List<String> inventory) { ❶
  System.out.println("Originator: Inventory set: " +
 inventory.toString());
  this.inventory = inventory;
 }

 public Memento saveStateToMemento() { ❷
  System.out.println("Originator: Saving inventory to Memento");
  return new Memento(inventory);
 }

 public List<String> getStateFromMemento(Memento memento) {❸
```

```
  inventory = memento.getInventory();
  System.out.println("Originator: Previously saved inventory of user :"
 + inventory.toString());

  return inventory;
 }
}
```

❶ Memento olarak kaydedilecek kendi nesnesini set eder.

❷ Yeni bir Memento oluşturup anlık envanteri ilgili Memento nesnesine atar.

❸ İlgili mementodan envanter bilgisini alır.

Memento.java.

```
public class Memento {
 private List<String> inventory = new ArrayList<>(); ❶

 public Memento(List<String> inventory) {
  this.inventory = inventory;
 }
 // getter and setters...
}
```

❶ Originator nesnesinin ilgili kopyası, Memento nesnesinde tutulacaktır.

CareTaker.java.

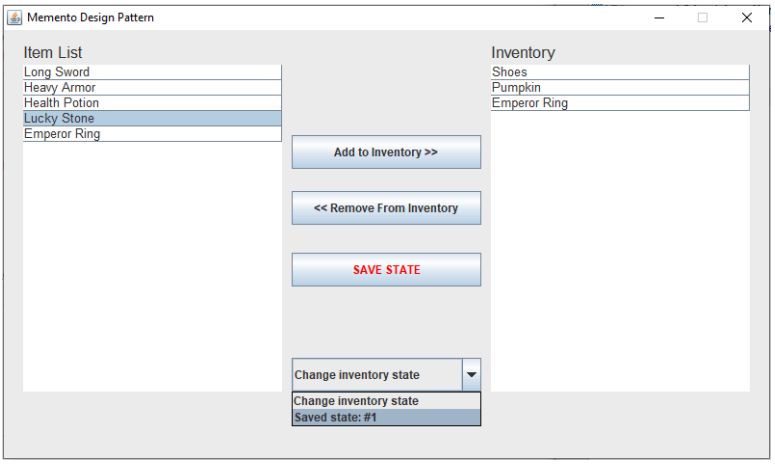
```
public class CareTaker {
 private List<Memento> mementoList = new ArrayList<Memento>(); ❶

 public void addMemento(Memento inventory) {
  mementoList.add(inventory);
 }
    // getters and setters...
}
```

❶ Tüm ilgili yedek nesneler, CareTaker sınıfı içerisindeki Memento listesinde tutulur.

1.4. Uygulama

- Uygulama arayüzünde bir eşya listesi ve kullanıcı envanteri tasvir edilmiştir.

- Kullanıcı listeden envantere eşya ekleyebilmekte ve envanterinden eşya silebilmektedir.
- İlgili değişikliklerden sonra **SAVE STATE** butonu ile **kullanıcı envanter durumu kaydedilir.**
    - Kaydedilen her state, Combobox'ta listelenmektedir.
    - İlgili kayda basılmak suretiyle hafızaya alınan envanter dizilimine erişilebilir.



Save butonu ve Memento nesnesinin oluşturulması

InventoryFrame.java.

```
btnSaveState.addActionListener(new ActionListener() {
 public void actionPerformed(ActionEvent e) {
  // save state for inventory

  // get current inventory items...
  List<String> inventoryList = new ArrayList<String>();

  // Set the value for the current memento
  originator.setInventory(inventoryList); ❶

  // Add new items the ArrayList
  caretaker.addMemento(originator.saveStateToMemento()); ❷

  currentInventory++;
```

```
  comboMemento.addItem("Saved state: #" + currentInventory); ❸
 }
});
```

❶ Kopyalanacak envanter, set edilir.

❷ Originator tarafından Memento nesnesi oluşturularak envanter kaydedilir. Ardından CareTaker içerisinde tutulan Memento listesine eklenir.

❸ İlgili kopya envanter kaydı, roll-back işlemi için combobox'a eklenir.

İlgili çıktı aşağıda verilmiştir.

```
InventoryFrame [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe  (3 Haz 2020 03:53:21)
Originator: Inventory set: [Shoes, Pumpkin, Emperor Ring]
Originator: Saving inventory to Memento
```

Nesnenin önceki duruma döndürülmesi, roll-back işlemi

InventoryFrame.java.

```
comboMemento.addActionListener(new ActionListener() {
 public void actionPerformed(ActionEvent e) {
  int stateNumber = Integer.parseInt(selectedCombo.replaceAll("[\
\D]", ""));

  List<String> inventoryList =
 originator.getStateFromMemento( caretaker.getMemento(--
stateNumber) ); ❶

 }
});
```

❶ Listeye eklenen numarası kullanılarak caretaker nesnesinden ilgili memento kaydı alınır.

Kodun çıktısı aşağıda verilmiştir:

```
Originator: Previously saved inventory of user :[Shoes, Pumpkin, Emperor
 Ring]
```