Memento (pamiątka, token)

Cel:

Nie naruszając hermetyzacji wydobywa i zachowuje wewnętrzny stan obiektu, tak by mógł być on przywrócony w późniejszym czasie.

Przykład:

```
class Punkt {
     private double x, y;
     public Punkt(double x, double y) { ... }
     public void przesuń(double dx, double dy) { ... }
}
class Client {
     public void main() {
          Punkt p = new Punkt(...);
          ...
     }
}
```

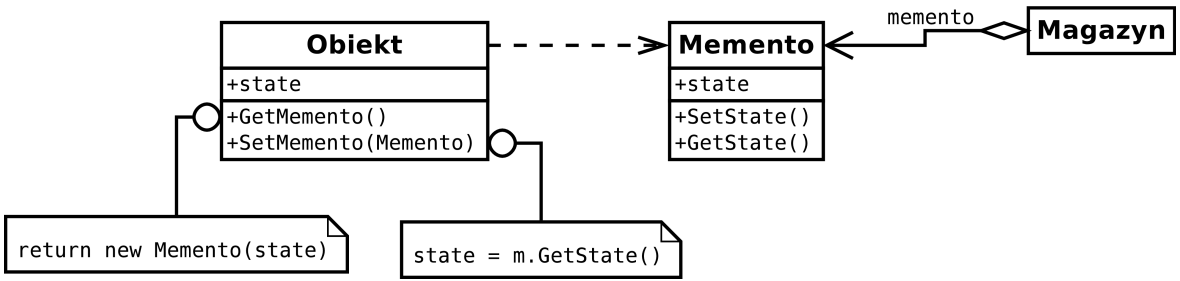
```
class Punkt {
     private double x, y;
     public Punkt(double x, double y) { ... }
     public void przesuń(double dx, double dy) { ... }
     public double getX() { ... }
     public double getY() { ... }
     public void setX(double value) { ... }
     public void setY(double value) { ... }
}
class Client {
     public void main() {
          Punkt p = new Punkt(...);
          ...
          // zapamiętujemy stan
          double[] tmp = new double[2];
          tmp[0] = p.getX();
          tmp[1] = p.getY();
          ...
          // odzyskujemy
          p.setX(tmp[0]);
          p.setY(tmp[1]);
     }
}
```

```
class Punkt {
     private double x, y;
     public Punkt(double x, double y) { ... }
     public void przesuń(double dx, double dy) { ... }
     private double mem_x, mem_y;
     public void save() { mem_x = x; mem_y = y; }
     public void load() { x = mem_x; y = mem_y; }
}
class Client {
     public void main() {
          Punkt p = new Punkt(...);
          ...
          // zapamiętujemy stan
          p.save();
          ...
          // odzyskujemy
          p.load();
     }
}
```

Zastosowanie:

- Gdy chcemy zachować stan obiektu, by móc go później odzyskać,
- a nie możemy uzyskać go wprost z obiektu, gdyż ujawni to implementację.

Struktura:



Składniki:

- Memento – przechowuje wewnętrzny stan Obiektu (a raczej tyle z tego stanu, ile Obiekt chce lub potrzebuje) i nie udostępnia go nikomu poza Obiektem.
- Obiekt – tworzy Memento przechowujące jego aktualny stan; może użyć je, aby go odzyskać.
- Magazyn – przechowuje memento, ale nigdy nie operuje na jego zawartości.

Zależności:

- Magazyn pobiera memento od Obiektu, składuje je przez jakiś czas, a następnie (w razie takiej potrzeby) znów mu odsyła.
- Dostęp do stanu memento ma tylko Obiekt, który je stworzył.

```
class Memento {
     ...
}

class Punkt {
     private double x, y;
     public Punkt(double x, double y) { ... }
     public void przesuń(double dx, double dy) { ... }
     public Memento getMemento() {
          return new Memento(x, y);
     }
     public void setMemento(Memento mem) {
          x = mem.getX();
          y = mem.getY();
     }
}
```

```
class Client {
     public void main() {
          Punkt p = new Punkt(...);
          ...
          // zapamiętujemy stan
          Memento mem = p.getMemento();
          ...
          // odzyskujemy
          p.setMemento(mem);
     }
}
```

Konsekwencje:

1. Zapobiega ujawnieniu informacji, o których powinien wiedzieć tylko właściciel, a które muszą być zapisane poza nim.
2. Upraszcza Obiekt – nie musi sam się troszczyć o przechowywanie swego stanu.
3. Może być kosztowne – jeśli Obiekt musi kopiować duże ilości danych lub klient często prosi go o memento. Jeśli zachowanie lub odzyskanie stanu jest zbyt drogie, lepiej może odnotowywać jedynie zmiany stanu.
4. Ukryte koszty – Magazyn nie wie jak duże jest memento, którym się zajmuje; w założeniu lekka klasa może okazać się kosztowna w składowaniu.

Implementacja:

1. Dwa interfejsy: szeroki dla Obiektu (źródła memento) i wąski (prawie zerowy) dla pozostałych. W Javie mogą do tego służyć klasy wewnętrzne.

```
interface Memento { }

class Punkt {
     ...
     private class PunktMemento implements Memento {
          double _x, _y;
          PunktMemento(double x, double y) { _x = x; _y = y;}
     }
     public Memento getMemento() {
          return new PunktMemento(x, y);
     }
     public void setMemento(Memento mem) {
          x = ((PunktMemento)mem)._x;
          y = ((PunktMemento)mem)._y;
     }
}
```

2. Przechowywanie przyrostowe zmian – jeśli memento powstają i wracają do Obiektu w przewidywalnej kolejności, zmiany można zachowywać przyrostowo („co się zmieniło od ostatniego razu”). Może to posłużyć do implementacji undo/redo.

```
interface Memento { }

class Matrix {
     private double[][] matrix;
     ...
     private class MatrixMemento implements Memento {
          int row, int col;
          double val_from, val_to;
     }
     public Memento set(int row, int col, double value) {
          Memento mem = new MatrixMemento();
          mem.row = row; mem.col = col;
          mem.val_from = matrix[row][col];
          mem.val_to = value;
          matrix[row][col] = value;
          return mem;
     }
}
```

3. W Javie do zapamiętania stanu obiektu można użyć *serializacji*.

```
class Moja implements Serializable {
     private int date;
     public Moja(int data) { this.data = data; }
     public String toString() { return "("+data+")"; }
}
public class Main {
     public static void main(String[] args) throws Exception
{
          Moja o = new Moja(100);
          ByteArrayOutputStream baos = new ByteArrayOutputStream();
          ObjectOutputStream oos = new ObjectOutputStream(baos);
          oos.writeObject(o);
          oos.close();

          ObjectInputStream ois = new ObjectInputStream(
               new ByteArrayInputStream(baos.toByteArray()));
          Moja odzyskany = (Moja) ois.readObject();
          ois.close();
          System.out.println(odzyskany);
     }
}
```

Powiązania:

- Command – może używać Memento do przechowywania zmian, których nie możemy łatwo cofnąć.
- Iterator – Memento również może być używane do iteracji: memento przechowuje stan iteracji, ale w odróżnieniu od iteratora przekazywane do kolekcji, by sięgnąć po następny element.