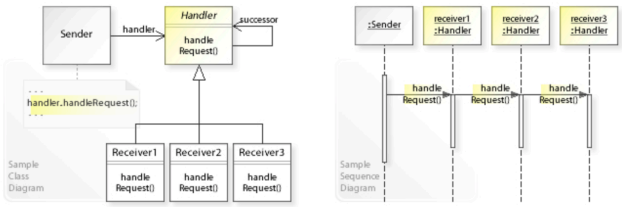
Behavioral design patterns:

Chain of Responsibility:



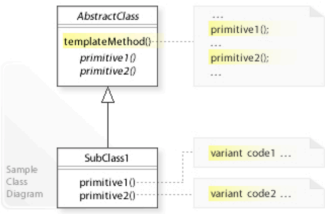
שימושי בעת צורך לביצוע פעולה על אוסייקט. כל פעולה, בהתאם לפרמטרים בזמן ריצה, מחליטה האם לפעול או להעביר את הטיפול לפעולה הבאה בשרשרת. בסופו של דבר רק פעולה אחת תפעל על האובייקט

מימוש: מגדירים מחלקה אבסטרקטית (handler) שמכילה אובייקט מהסוג של המחלקה, ומכילה פונקציה אבסטרקטית (handleRequest) שאחראית על הפעלת הפעולה הנוכחית, וקריאה לפעולה הבאה בתור.

חסרונות:

- יש להימנע מפעולות מעגליות, למשל מצב כזה: reciver1 -> reciver2 -> reciver1 במצב כזה הלולאה לא תעצר.
- אם אחד מהפעולות בדרך נכשלת, הרצף כולו נקטע.

Template Method:

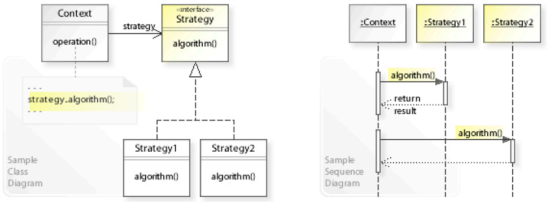


שימושי כאשר נרצה לבנות פעולות שכדי לבצע אותן צריך לעשות מספר צעדים **קבועים** אחרי או לפני, למשל, כתיבה ל db: יש ליצור חיבור לdb, לבצע את הפעולה ולהמיר את המידע או במקרה של כישלון יש לעשות rollback.

במצב כזה הפוקציה הראשית במחלקה תבצע את הפעולות הקבועות, והפעולה המשתנה תהיה אבסטרקטית. בדוגמא למעלה הפעולות האבסטרקטיות הן primitive1() ו- primitive2().

ב Java ישנם מספק אוספים שמשתמשים ב Template למשל AbstractList ו- AbstractSet. בדוגמא של AbstractList, בפונקציה addAll, על המשתמש לממש רק את הפונקציה add(index, element), וכל שאר הלוגיקה ממומשת.

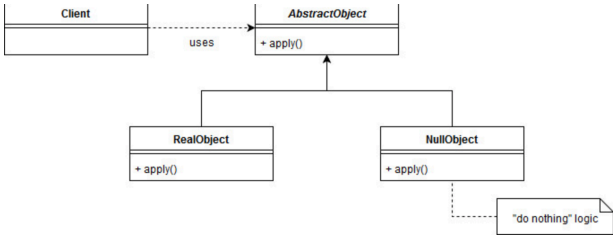
Strategy:



תבנית שמאפשרת החלפת מימוש אלגוריתם בזמן ריצה.
מימוש: מגדירים מחלקה אבסטרקית/ממשק שמכיל פונקציה עם אלגוריתם מסויים. ממשמשים את הממשק במספר דרכים, ובכך בזמן ריצה אפשר לבחור איזה מימוש נרצה להשתמש בו.
מידע נוסף: ניתן להגדיר על ידי java 8 ממשק שיאפשר קומפוזיציה של פעולות (כלומר הפעלת פעולה על פעולה) כך:

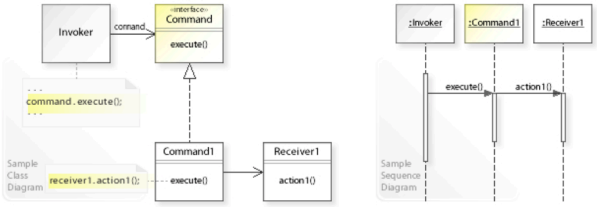
```
public interface Discounter extends UnaryOperator<BigDecimal> {
    default Discounter combine(Discounter after) {
        return value -> after.apply(this.apply(value));
    }
}
```

Null Object: מקרה מיוחד של Strategy



הלוגיקה של התבנית הזו היא במקום שהמשתמש יבדוק האם האובייקט הוא null, נגדיר null object של הממשק/מחלקה אשר לא יעשה כלום. ובכך לא נצטרך טיפול מיוחד ל null.
אין צורך ליצור את האובייקט null כל פעם שנצטרך אותו, לכן נכון להשתמש ב singleton.

command:



תבנית המאפשרת ביצוע פעולות שונות על אובייקט.

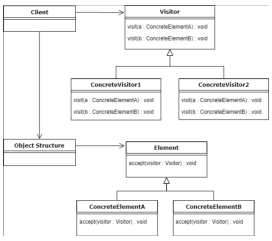
מימוש:

- command: הממשק שמייצג את הפעולה שניתן לעשות על האובייקט. כל פעולה (command) יממש את הממשק (למשל מחלקה לפתיחת קובץ ולסגירת קובץ). מימוש לדגומא: textfile.open()
- receiver: האובייקט עליו עובדים. המימוש האמיתי של ה commands יהיה באובייקט הזה. מימוש לדוגמא: המימוש עצמו של open() לדוגמא פתית הקובץ וכתיבה ללוג.
- invoker: המחלקה שקוראת לפונקציות של ה command. המחלקה לא יודעת איך ה command ממומש. לרוב המחלקה מכילה רשימה של הפעולות שהתבצעו על האובייקט, דבר שמאפשר הצגת היסטוריה של פעולות, ביטול פעולה וכו.

ההבדל בין command ל - Strategy:

- משתמשים ב command כאשר נרצה להגיד מה נעשה על אובייקט. בכך נוכל לשמור היסטוריה של פעולות, לבטל וכו. דוגמא: פתיחת קובץ, עריכת קובץ, סגירת קובץ, שינוי שם קובץ וכו.
- משתמשים ב strategy כאשר נרצה לציין איך משהו יעבוד, לדוגמא לאפשר אלגוריתמים שונים לעבודה על אובייקט. דוגמא: אלגוריתמים שונים של מיון.

Visitor:



תבנית שיעילה כאשר נרצה לעבד data structure המכיל סוגים שונים של אובייקטים. פעולת העיבוד תפעל שונה על כל סוג של אובייקט, ובכך נוכל לעבד את הישות כולה. התבנית עונה על הקריטריון של open/closed, כי כאשר נרצה להוסיף התנהגות, כל שנצטרך הוא להוסיף מחלקה שתממש את הממשק visitor, כלומר תראה כיצד היא תפעל על כל סוג אובייקט.

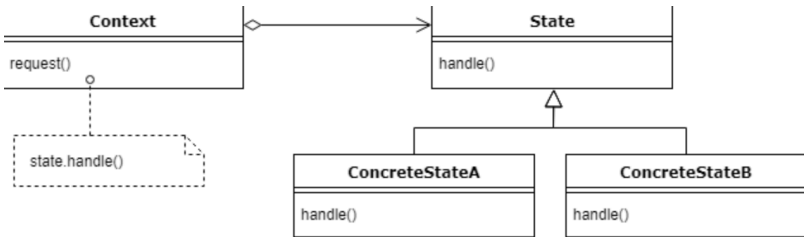
מימוש:

- visitor: ממשק שמגדיר איך עליו לפעול על כל סוג אובייקט. הפעולה תהיה עם השם visit(Element e).
- ConcreteVisitor: האובייקט שיממש את הממשק של visitor. במימוש עליו להראות כיצד יפעל על כל אחד מהאובייקטים.
- object structure: האובייקט שיכיל סוגים שונים של Elements
- Element: ממשק שמייצג את האובייקטים השונים. הממשק יכיל פעולה של כיצד האובייקט התנהג כשאר נפעיל עליו את הפעולה של הvisitor.
- ConcreteElement: המחלקה שתממש את הממשק Element.

חיסרונות:

- כאשר נרצה להוסיף סוג חדש של אובייקט (לדוגמא מערכת הפעלה ios), נצטרך לשנות את החתימה של ה visitor שידע לפעול על האובייקט החדש, בנוסף נשנה את הConcreteVisistors שידעו לטפל באובייקט החדש.

State:

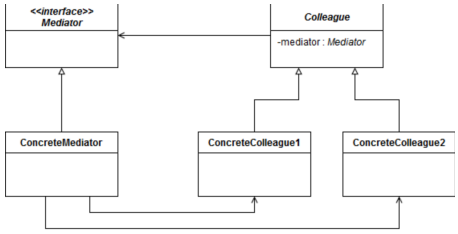


תבנית שמאפשרת טיפול שונה לאובייקט בהתאם למצב state בו הוא נמצא, ומעבר קל בין המצבים השונים.
התבנית עונה על הקריטריון של Open/Closed ו Single Responsibility. אם נרצה להוסיף מצב חדש לאובייקט, נצטרך רק לממש את State.
מימוש:

- State: ממשק שמגדירה מצב של אובייקט. המצב מכיל את הפעולה אותה נרצה לבצע במצב זה, המעבר למצב הבא והמעבר למצב הקודם.
- ConcreteState: מחלקה שמממשת את State, ומייצגת מצב מסויים של האובייקט.

ספרייה: בפרוייקט של spring boot יש את Spring state machine שמממש מכונת מצבים.

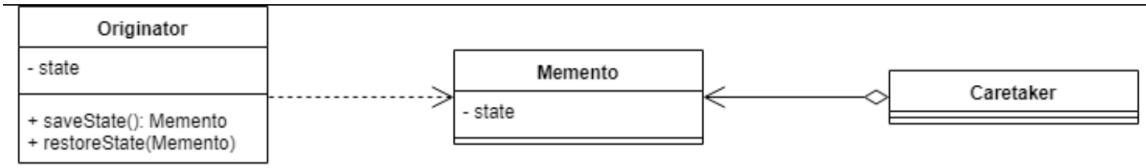
Mediator:



תבנית שימושית כאשר יש תלות גבוה בין אובייקטים שונים במערכת.
התבנית נועדה כדי למנוע תקשרות ישירה בין האובייקטים, ולעומת זאת לבצע תקשורת דרך אובייקט משותף Mediator.
המחלקה עונה על הקריטריון של Single Responibility, מכיוון שהתפקיד שלה הוא רק לקשר בין האובייקטים השונים. בנוסף ניתן להחליף את ה Mediator במימוש אחר כרצוננו ולכן זה ממשש את Open/Closed.
דוגמא: במכונת קירור, כאשר נלחץ על **הכפתור** צריך להדליק את **החשמל** ואז את **המאוורר**. לכן לאובייקט כפתור יהיה שדה של מאוורר שבתוכו יחזיק אובייקט מקור אנרגיה. במצב כזה יש תלות גבוה מדי בין האובייקטים במערכת.
מימוש:

- Mediator: ממשק שמייצג אובייקט שמקשר בין אובייקטים שונים.
- ConcreteMediator: המחלקה שמממשת את Mediator, ומכילה את התקשורת בין האובייקטים השונים (כלומר היא תחזיק את האובייקטים שתלויים אחד בשני).
- Colleague: האובייקט במערכת. האובייקט יחזיק שדה Mediator שיאפשר לו לתקשר עם שאר האובייקטים.

Memento:



תבנית שמממשת שמירת מצב קיים של אובייקט, וחזרה למצב שמור. דוגמא text editor.

מימוש:

- Originator: המחלקה שמחזיקה את האובייקט שנרצה לשמור לו את המצב שלו. כאשר שומרים את המצב של האובייקט המחלקה מייצרת Memento חדש דוגמא TextWindow.
- Memento: מחלקה שמייצגת את המצב השמור TextWindowState.
- Caretaker: המחלקה שמנהלת את המצבים של האובייקט, היא מכילה את הפונקציות של לישמור את המצב ולהחזיר לאחור. מכילה את ה Memento. דוגמא TextEditor.

חסרונות:

- כאשר נצטרך לשמור מצב מערכת כבד מידי, יכול להיות בעיה של זכרון