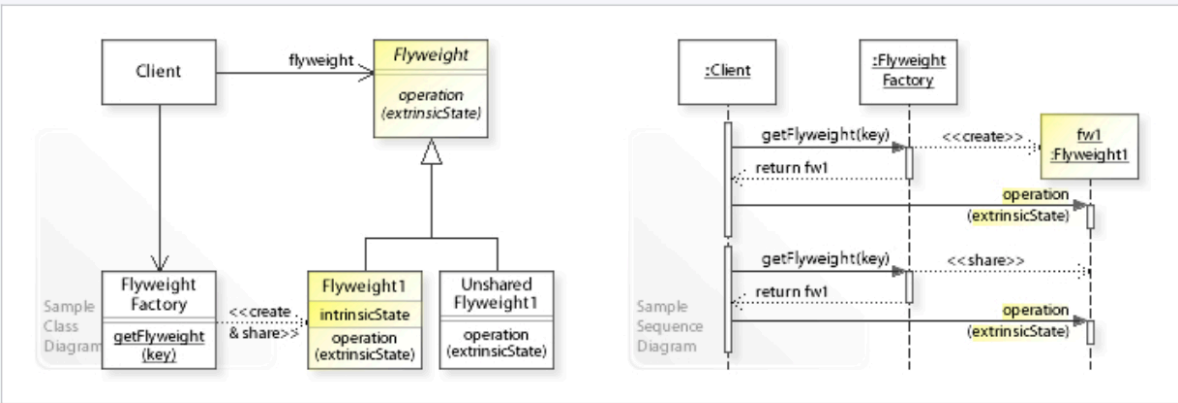
Structural design patterns:

Flyweight:



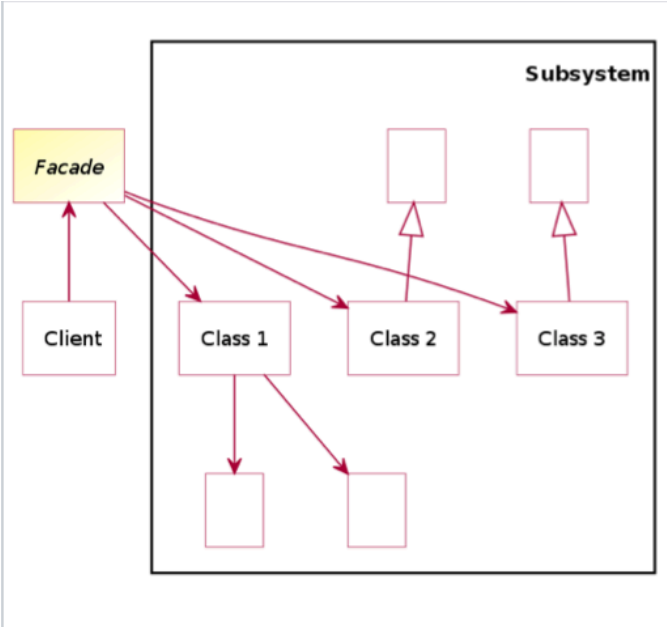
תבנית שימושית כשאר נרצה למחזר אובייקטים.

מימוש:

- Element: ממשק שבו יוגדר הפעולות שהמשתמש יוכל לעשות עם האובייקטים.
- ConcreteElement: מימוש של הממשק
- Factory: המחלקה שתהיה אחראית על יצירת האובייקטים וה caching

מימוש עם java 8: כאשר נשתמש בפונקציה Map.computeIfAbsent נוכל להחזיר את האובייקט אם הוא קיים, אחרת לחשב אותו בפעם הראשונה והלחזיר אותו.

Facade:

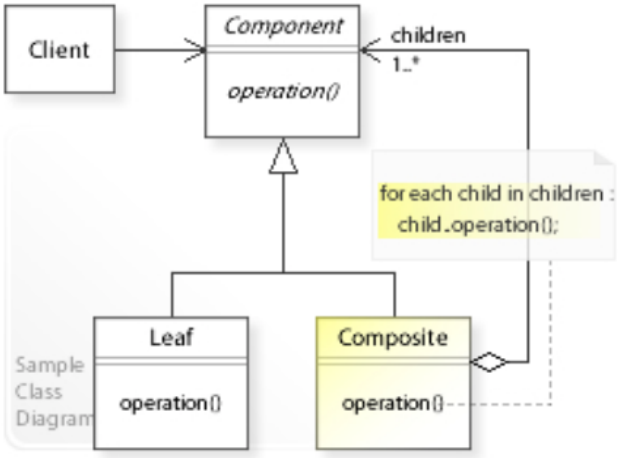


כאשר במערכת ישנן פעולות מורכבות שמכילות סאב פעולות רבות, כמו בדוגמא למטה:

```
airFlowController.takeAir()
fuelInjector.on()
fuelInjector.inject()
starter.start()
coolingController.setTemperatureUpperLimit(DEFAULT_COOLING_TEMP)
coolingController.run()
catalyticConverter.on()
```

במצב כזה, ניתן לפשט את הApi על ידי הוספת רמת אבסטרקציה בכך שנכניס את כל הסאב פעולות הללו לפונקציה אחת. ובכך נסתיר את המורכבות מהמשתמש.
בדוגמא למעלה: נכניס את כל הפעולות לפונקציה ()startEngine.

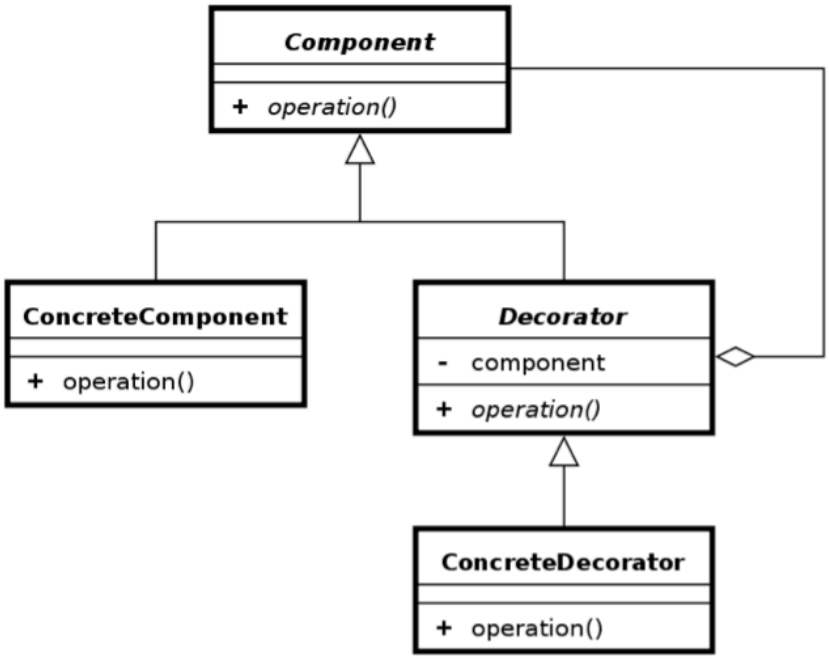
Composite:



תבנית שמאפשרת טיפול באובייקט פשוט ובגרסה המורכבת שלו באותו אופן.
מימוש:

- Component: ממשק
- Leaf: אובייקט שמממש את הממשק Component.
- Composite: אובייקט שמממש את הממשק Component ומכיל רשימה של אובייקטים מסוג Compoennt.

Decorator:



תבנית שמאפשרת להרחיב התנהגות/תכונות של אובייקט בזמן ריצה או בעת יצירת האובייקט.
התבנית עונה על הקריטריון של Single Responsibility

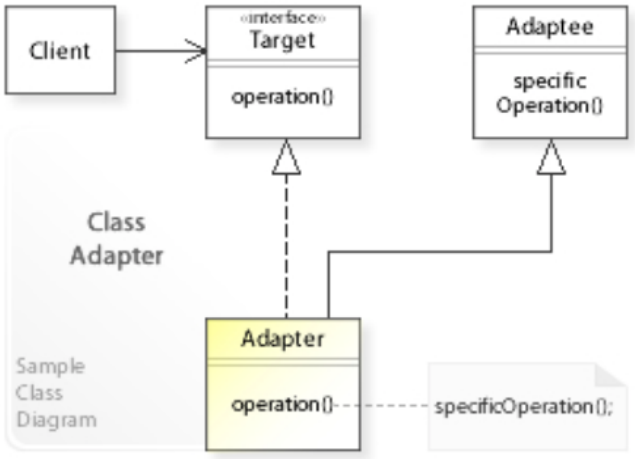
מימוש:

- Component: ממשק שמגדיר את הפעולה של האובייקט.
- ConcreteComponent: האובייקט שמממש את הממשק Component. אובייקט הבסיס.
- Decorator: מחלקה אבסטרקטית שמחזיקה שדה מסוג Component וממשת את הממשק Component.
- ConcreteDecorator: מחלקה שמרחיבה את ה Decorator, ובמימוש של הפונקציה של הממשק היא מרחיבה את המימוש של אובייקט אותו היא מכילה.

נקודות חשובות:

- למרות שProxy דומה ל Decortor, הם נבדלים בכך ש Proxy מסתיר מורכבות של אובייקט, בעוד Decorator מוסיף סמכויות/התנהגויות לאובייקט.
- Proxy, Decorator and Adapter שלושם מחזיקים reference לאובייקט האמיתי.
- כל ה הConcreteDecorators יכולים להיות בשימוש מספר אין סופי של פעמים (בצורה רקרוסיבית), בשונה משאר הpatterns.

Adapter:



תבנית שבאה לקשר בין שני ממשקים שלא מתאימים באופן ישיר. ה Adapter עוטף את המחלקה עם ממשק חדש שהמשתמש מצפה לעבוד איתו.
מתאים כאשר עובדים עם ספריות חיצוניות ורוצים להוריד את התלות של הקוד בממשק של הספרייה, כדי שנוכל בקלות להחליף אותה.

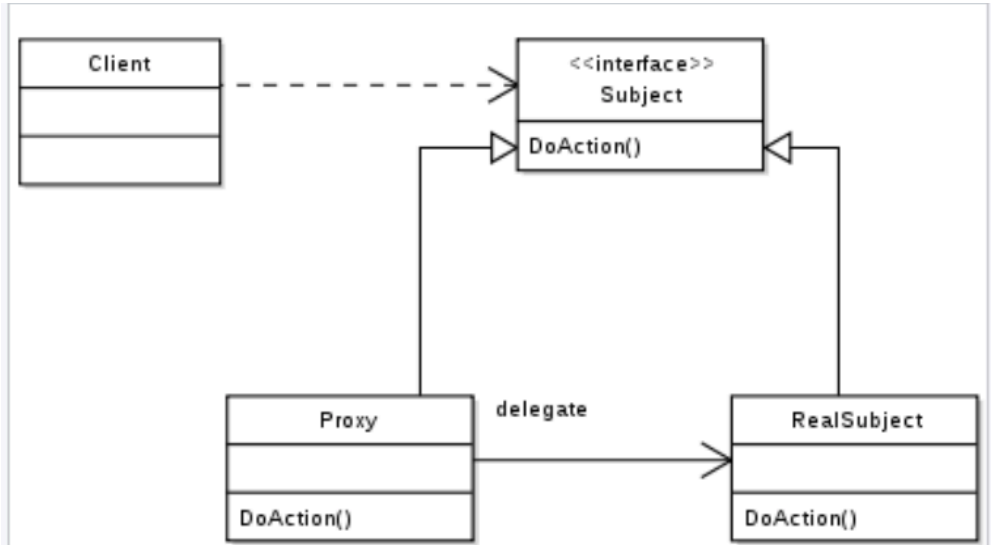
מימוש:

- Traget: הממשק שהמשתמש מכיר.
- Adaptee: האובייקט שנרצה לעטוף ולשנות את ההתנהגות שלו.
- Adapter: המחלקה שמכילה את המימוש של הפעולה.

נקודות חשובות:

- בעוד ש Proxy מממש את הממשק המקורי של האובייקט, ה Adapter מממש ממשק שונה, כזה שיתאים יותר לשימוש של המשתמש.
-

Proxy:



תבנית שנועדה להוסיף אבסטרקציה לאובייקט כדי להסתיר מורכבות שלו. כך ניתן לפתור בעיות כגון:

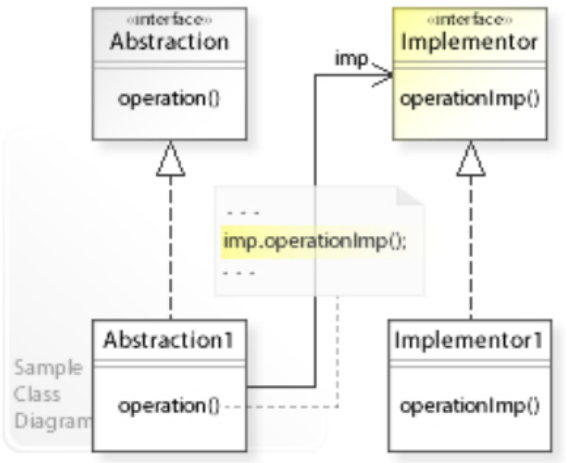
- אתחול lazy של אובייקט שדורש איתחול כבד.
- התחייסות לאובייקט מרוחק כאילו היה מקומי. הפתרון יהיה להסתיר את

המורכבות של התקשורת וכו מהמשתמש.
- הוספת רמה של אבטחה על אובייקט

מימוש:
- Subject: ממשק שמוכר למשתמש.
- RealSubject: אובייקט שמממש את ה ממש Subject.
- Proxy: אובייקט שמממש את הממשק Subject. ה Proxy מחלקה מסוג RealObject, וכך הוא מוסיף שכבה נוספת על האובייקט.

נקודות נוספות:
- ה proxy מממש את הממשק של האובייקט אותו הוא מחזיק. הוא לא משנה את האובייקט בשום צורה, לא כמו הAdapter ו Decorator.
- ה proxy מכיל אובייקט שידוע בזמן קומפילציה, לעומת decorator ו Adapter שמרכיבים את האובייקט בזמן ריצה.

Bridge:



תבנית שנועדה להפריד תלויות בין אובייקט לאובייקט אחר המוכל בתוכו, כך ששני האובייקטים יכולים להשתנות באופן בלתי תלוי אחד בשני.
דוגמא: מחלקה אבסטרקטית של Shape שמכילה אובייקט מסוג Color. ניתן לממש את Shape לכל צורה שנרצה כגון Circle ו Triangle, ובאופן בלתי תלוי נוכל לממש את Color לכל צבע שנרצה כגון Red ו Green.

מימוש:
- Abstractation: הממשק המייצג את ההתנהגות של האובייקט הראשון.
- Abstractation1: מחלקה שמממשת את הממשק, המחלקה מכילה אובייקט מסוג Implementor
- Implementor: הממשק שמייצג את ההתנהגות של האובייקט השני.
- Implementor: מחלקה שמממשת את הממשק השני.

נקודות נוספות:
- מאפשר לאבסטרקציה ולמימושים להתשנות באופן בלתי תלוי.