Bridge (most)

Cel:

Oddzielenie abstrakcji od implementacji, tak aby mogły zmieniać się niezależnie. Zazwyczaj robi się to przez dziedziczenie, ale komplikuje to hierarchię.

Przykład:

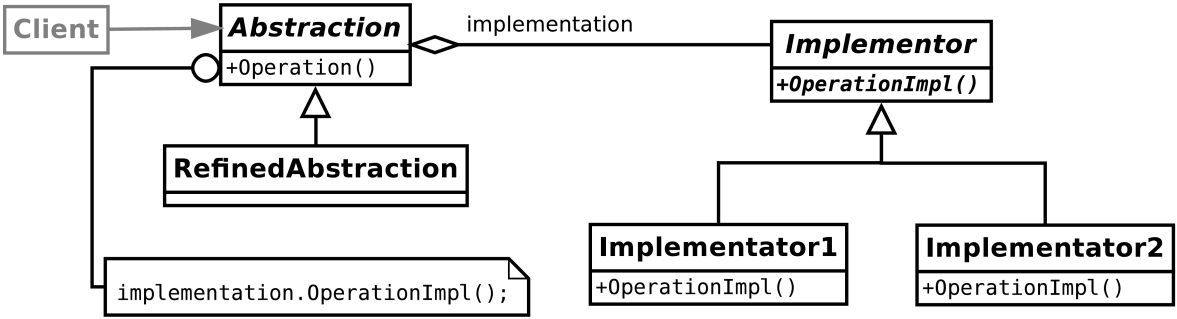
```
interface Figure {
     void draw();
     // ...
}
class Triangle implements Figure{
     // ...
}
class Oval implements Figure{
     // ...
}
class Rectangle implements Figure{
     // ...
}
```

```
interface Figure {
     void draw();
     // ...
}
class Triangle2D extends Triangle {
     // ...
}
class Oval2D extends Oval {
     // ...
}
class Rectangle2D extends Rectangle {
     // ...
}
class Triangle3D extends Triangle {
     // ...
}
class Oval3D extends Oval {
     // ...
}
class Rectangle3D extends Rectangle {
     // ...
}
```

Zastosowanie:

- aby zapobiec trwałemu związkowi między abstrakcją i implementacją, np. gdy chcemy wybrać implementację w trakcie działania programu
- gdy zarówno abstrakcja, jak i implementacja powinny być rozszerzalne przez dziedziczenie; bridge pozwala rozszerzać je niezależnie od siebie
- gdy zmiany w implementacji abstrakcji nie powinny wpływać na klienta, ani wymagać rekompilacji jego kodu
- gdy chcemy całkowicie ukryć implementację jakiejś abstrakcji przed klientem
- jeśli mamy mocno rozgałęzioną hierarchię przydaje się rozdzielić obiekt na dwie części (zagnieżdżona generalizacja)
- aby bez wiedzy klienta współdzielić implementację między różnymi obiektami

Struktura:



Składniki:

- Abstrakcja (**Abstraction**)
  - przechowuje referencje na obiekt Implementora
- Implementacja (**Implementor**)
  - definiuje interfejs klas implementujących – posługuje się nim abstrakcja; oczywiście nie muszą (i przeważnie nie są) to takie same interfejsy – implementacja raczej ma operacje niskiego poziomu, a abstrakcja wysokiego;

Zależności:

Abstrakcja oddelegowuje żądania klienta do Implementacji.

```
interface Figure {
     void draw();
     // ...
}

abstract class AbstractFigure implements Figure {
     protected Implementor impl;
     public AbstractFigure(Implementor impl) {
          this.impl = impl;
     }
}

interface Implementor {
     void drawLine(int Point, int Point);
}
```

```
class Triangle extends AbstractFigure{
     public Triangle(Implementor impl) {
          super(impl)
     }
     public void draw() {
          for(/* ... */)
               impl.drawLine(/* ... */);
     }
     // ...
}

class Oval extends AbstractFigure { /* ... */ }
class Rectangle extends AbstractFigure { /* ... */ }

class Implementor2D implements Implementor {
     public void drawLine(int Point, int Point) {
          // ...
     }
}
class Implementor3D implements Implementor {
     // ...
}
```

Konsekwencje:

1. oddziela interfejs od implementacji – nie ma już między nimi stałego związku, można nawet zmienić je w locie; nie ma też konieczności rekompilacji jednego modyfikując drugie; wymusza to rozwarstwienie systemu, co prowadzi do lepszego projektu
2. można niezależnie rozszerzać abstrakcję i implementację
3. ukrycie szczegółów implementacji przed klientem – np. współdzielonego implementora, czy liczników referencji

Powiązania:

- Abstract Factory może stworzyć i skonfigurować Bridge
- Adapter też ma łączyć różne klasy o różnych interfejsach, ale stosuje się go raczej do klas już gotowych, a Bridge jest używany w momencie projektowania systemu