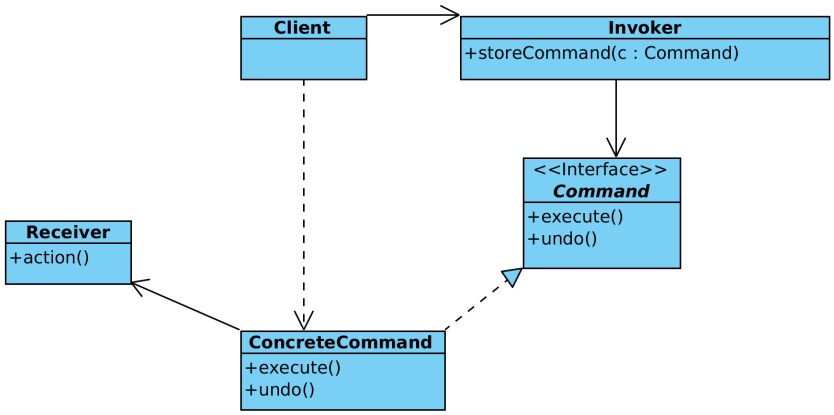
# 1 Komenda

Zawieramy polecenie w osobnym obiekcie.

**Elementy i połączenia** :

- Sterujący/klient - tworzy obiekty komend i przekazuje je do późniejszego wywołania, może być ich wielu, np. również inne komendy.
- Interfejs komendy.
- Konkretna komenda - obiekt klasy zawiera wywołanie pewnego podprogramu.
- Wykonujący - wykonuje komendę w odpowiednim czasie.
- Niekiedy wyróżnia się też obiekt, na którym komenda wywołuje operacje (zakładając przy tym, że powinien być jeden).




Rysunek 1: Struktura klas Polecenia

Ogólnie:

```
public interface class Command{
  setParameters(/***/);
  execute();
  unexecute();
}
```

```
public class ConrecteCommand implements Command{
  setParameters(/***/){
  /*...*/
  }
  execute(){
    realReceiver.doAction(storedParameters);
  }
  unexecute(){ //Opcjonalna
    /*...*/
  }

}

public class Invoker(){
Queue<Command> qc;
  enqueue(Command c){
    qc.add(c);
    qc.notify();
  }
  run(){
    while(!shouldStop()){
      while(!qc.empty()){
qc.pop().do();
      }
      wait(qc);
    }
  }
}
```

trochę bardziej specyficznie:

```
public interface class Editor{
  execute();
  undo();
}

public class DeleteCommand{
  String deleted;
  Postion from;
  Postion to;
  /*getter setter*/
  execute(){
```

```
    deleted= document.deleteCharacters(from,to);

  }
  undo(){
    document.insertCharacters(from,deleted);
  }

}

public class DeleteCommandHandler{
  deleteActionSelected(){
    Position from = getSelection().from();
    Postion to = getSelection().to();
    editor.executeCommand(new DeleteCommand(from,to));
  }
}
public class Editor(){
  Stack<Command> cs;
  undoLast(){
    c.undo();
    cs.pop();
  }
  executeCommand(Command c){
    c.execute();
    cs.push(c);
  }
}
```

- Przydatne jeśli formułowanie (specyfikacja) kolejkowanie i wykonywanie poleceń odbywają się w różnym czasie (lub w różnych miejscach kodu), np. chcemy przechowywać polecenia (np. na dysku, albo tyczasowo) przesyłać je, lub wspierać "odczynianie" poleceń. Żadne z tych dodatkowych zachowań nie jest wymagane przy wzorcu komendy, ale jest ona środkiem ich realizacji.
- Może być odpowiednikiem metod "callback", ale przechowuje też stan.
- Komenda usuwa sprzężenie pomiędzy obiektem, który wydaje polecenie, a tym, który wie, jak je wykonać.
- Polecenie może być obiektem "się" wykonującym - czyli zawierającym nietrywialną logikę, albo prostym przekazaniem wywołania. Zazwyczaj przynajmniej część operacji deleguje.

- Dodawać polecenia można łatwo - bez zmiany dotychczasowej infrastruktury poleceń.
- Można tutaj sporo dodać - np. nagrywanie i odtwarzanie makr, pewien rodzaj skryptowania, wysyłanie strumienia poleceń.
- Strategia może mieć trochę podobną strukturę, ale komenda zasadniczo wyraża enkapsulację wywołania algorytmu, a nie samego algorytmu (np. ważniejszy jest tu stan obiektu) i rozwiązuje trochę inny problem. Wywołujący ją zwykle mniej zakłada o tym, co zostanie zrobione.
- Niekiedy implementowalne przez "closure".

**Zadanie 1.** Proszę napisać "ręcznie" prosty edytor tekstowy udostępniający stos wycofywania oraz "odwycofania" (ponawiania) za pomocą wzorca komendy. Powinien mieć przynajmniej 2 rodzaje podstawowych poleceń - wpisywanie i usuwanie (backspacem), i oba powinny się dać "odwrócić". Nie musi być GUI, choć tak może być nawet łatwiej. Najważniejszą częścią jest komenda - czyli enkapsulacja wywołań i oddzielenie utworzenia od rzeczywistego wywołania. Powinny raczej być dwie komendy - wpisywanie i wycofywanie, każda ma zaimplementowane "zrób" i "wycofaj" (nie należy używać jednej komendy jako wycofywanie drugiej).

# 2 Dekorator (decorator)

. W skrócie: używamy kompozycji zamiast tworzenia podklas do zmiany zachowania obiektu (przede wszystkim do elastycznego dodania nowej odpowiedzialności) Wciąż używamy dziedziczenia interfejsu.

**Elementy i połączenia** :

- Obiekt dekorowany, przechowywany przez dekorator, jego metody są zwykle wywoływanie jako część wywołań metody dekoratora.
- Dekorator - przechowuje dekorowany obiekt, dostarcza zasadniczo takiego samego interfejsu, ale dodaje coś do operacji (zwykle wywołuje operacje obiektu dekorowanego).

Kod wygląda mniej więcej tak:

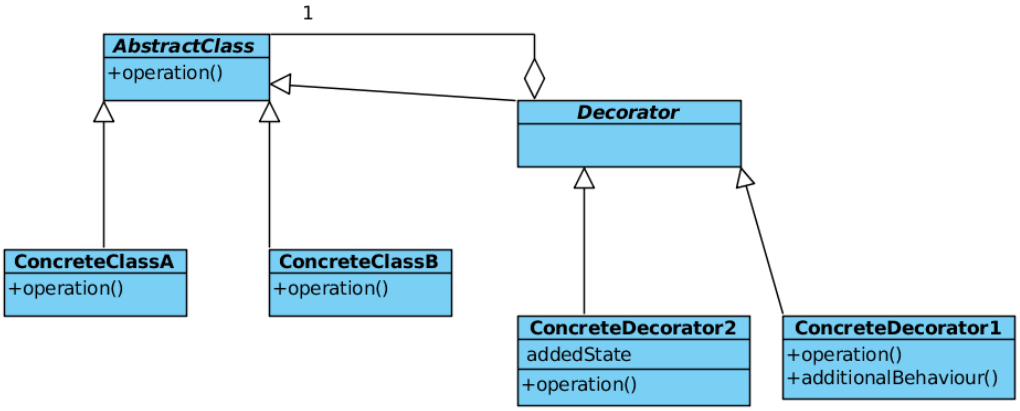
```
1 interface Dekorowany{
    operacja();
3 }
```

```
class DekorowanyKonkretnyA implements Dekorowany{
  operacja(){
   //...
   }
}
class DekorowanyKonkretnyB implements Dekorowany{
  operacja(){
   //...
   }
}
class Dekorator implements Dekorowany{
    Dekorowany dekorowany;
    operacja(){
      // Zrób coś jeszcze...
      dekorowany.operacja();
      // Zrób coś innego
    }
    Dekorator(Dekorowany d){
      this.dekorowanyu = d;
    }
}

class KlientDekoratora{
//...
  zrobCos(){
  //...
    Dekorator dr = new Dekorator(new
   DekorowanyKonkretnyA());
    Dekorator dr2 = new Dekorator(new
   DekorowanyKonkretnyB());
    dr.operacja();
    dr2.operacja();
  }
}
```

Nieco bardziej konkretnie to np.:

```
abstract class GraphicsObject{
  paint();
}
class Button extends GraphicsObject{
```




Rysunek 2: Strutkura Dekoratora

```
    paint(){
      // paint a button
      }
}
class TextField extends GraphicsObject{
    operacja(){
      // paint a text field
      }
}
class GracphisObjectWithBorder extends GraphicsObject{
      GraphicsObject primary;
      operacja(){
    paintBorders(primary);
    primary.paint();
      }
}
```

- Dekorator raczej nie powinien wymagać informacji, jaki dokładnie obiekt dekoruje, tylko polegać na abstrakcji - interfejsie. Można stosować wtedy dekorowanie wielopoziomowe.
- Można wymieniać dekoratory niezależnie od klas dekorowanych – duża elastyczność.
- Można stosować dekorację wielopoziomową.

- Można zrobić dużo rzeczy, nawet całkowicie nadpisać metodę, jednak mamy ograniczony dostęp do wewnętrznego stanu obiektu dekorowanego (tyle, na ile pozwala interfejs) i nie możemy ingerować w wykonanie metody obiektu bazowego (np. jeśli tam była template method) - dlatego możliwości zmian są mniejsze niż np. w dziedziczeniu.

**Zadanie 2.** Proszę napisać do poprzedniego zadania dekoratory komend, które zliczą liczbę komend które utworzyły obecny tekst (przy wycofywaniu - odejmują).