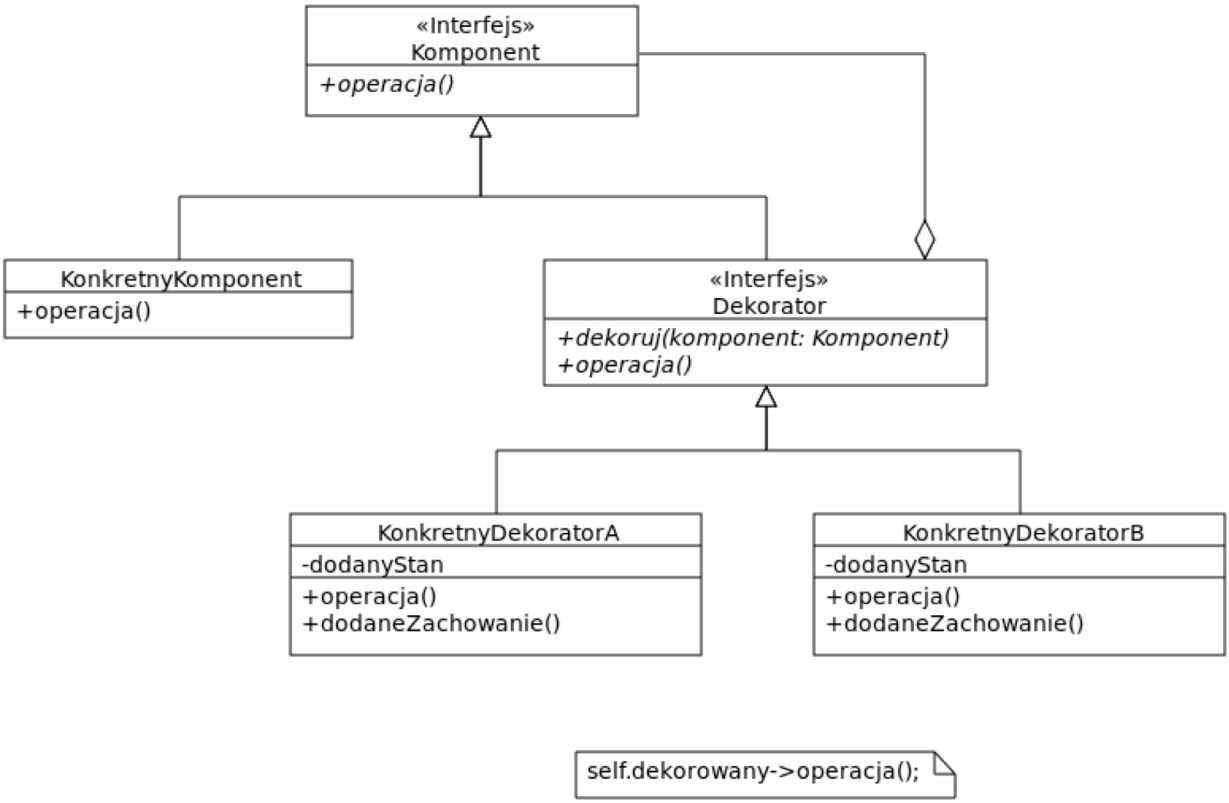
Inżynieria oprogramowania

Wzorce projektowe: dekorator

1. Wzorzec należy do grupy wzorców strukturalnych i pozwala na:
   - dynamiczną modyfikację klas podczas działania jako alternatywa dziedziczenia,
   - rozwiązuje problem braku możliwości tworzenia nowych klas podczas działania programu,
   - wyklucza problem tworzenia wszystkich podzbiorów klas pochodnych (różnych ich kombinacji)

2. Schemat UML dekoratora:



3. Proszę przeanalizować działanie poniższego kodu po czym przepisać go i sprawdzić jego działanie:

```
// interfejs Okno
interface Okno {
    public void rysuj(); // rysuje Okno na ekranie
    public String pobierzOpis(); // zwraca opis Okna
}

// implementacja zwykłego okna bez pasków przewijania
class ZwykłeOkno implements Okno {
    public void rysuj() {
        // rysuj okno
    }

    public String pobierzOpis() {
        return "zwykłe okno";
    }
}
```

Poniższe klasy zawierają dekoratory dla wszystkich klas Okno, w tym też dla klas dekoratorów.

```
// abstrakcyjna klasa dekorator - implementuje interfejs Okno
abstract class OknoDekorator implements Okno {
    protected Okno dekorowaneOkno; // dekorowane Okno

    public OknoDekorator(Okno dekorowaneOkno) {
        this.dekorowaneOkno = dekorowaneOkno;
    }
}

// pierwszy dekorator dodający pionowe paski przewijania
class PionowePrzewijanieDekorator extends OknoDekorator {
    public PionowePrzewijanieDekorator (Okno dekorowaneOkno) {
        super(dekorowaneOkno);
    }

    public void rysuj() {
        rysujPionowyPasekPrzewijania();
        dekorowaneOkno.rysuj();
    }

    private void rysujPionowyPasekPrzewijania() {
        // rysuj pionowy pasek przewijania
    }
```

```
    public String pobierzOpis() {
        return dekorowaneOkno.pobierzOpis() + ", z pionowym paskiem przewijania";
    }
}

// drugi dekorator dodający poziome paski przewijania
class PoziomePrzewijanieDekorator extends OknoDekorator {
    public PoziomePrzewijanieDekorator (Okno dekorowaneOkno) {
        super(dekorowaneOkno);
    }

    public void rysuj() {
        rysujPoziomyPasekPrzewijania();
        dekorowaneOkno.rysuj();
    }

    private void rysujPoziomyPasekPrzewijania() {
        // rysuj poziomy pasek przewijania
    }

    public String pobierzOpis() {
        return dekorowaneOkno.pobierzOpis() + ", z poziomym paskiem przewijania";
    }
}
```

Program testowy, który tworzy obiekt klasy Okno, dekoruje go poziomymi i pionowymi paskami przewijania i wypisuje jego opis.

```
public class DekorowaneOknoTest {
    public static void main(String[] args) {
        // utwórz dekorowane Okno z poziomymi i pionowymi paskami przewijania
        Okno dekorowaneOkno = new PoziomePrzewijanieDekorator(
                new PionowePrzewijanieDekorator(new ZwykłeOkno()));

        // wypisz opis Okna
        System.out.println(dekorowaneOkno.pobierzOpis());
    }
}
```

4. Przykład kolorowania napisów:

```
public static final String ANSI_RED_BACKGROUND = „\u001B[41m”;
public static final String ANSI_GREEN = „\u001B[32m";
public static final String ANSI_RESET = „\u001B[0m";

public static void main(String[] args) {
      System.out.print(ANSI_RED_BACKGROUND+”test!”+ ANSI_RESET);
      System.out.print(ANSI_GREEN+”test!”+ ANSI_RESET);
```

```
}
```

5. Korzystając z opisywanego wzorca projektowego i wzorując się na powyższym przykładzie proszę stworzyć program rzeczywiście dekorujący napisy na konsoli:
   (1) stworzyć interfejs o nazwie `Napis` zawierający metodę `wypisz()`, która będzie używana do wypisania napisu na konsolę;
   (2) stworzyć klasę `KrotkiNapis` implementującą interface (1) zawierającą odpowiednie pole typu String;
   (3) stworzyć klasę abstrakcyjną `DekoratorNapisu` implementującą interface (1) i zawierającą pole chronione `dekorowanyNapis` typu `Napis`;
   (4) stworzyć klasę `CzerwoneTloDekorator` dziedziczącą po `DekoratorNapisu` . Do pokolorowania napisu proszę użyć metody z poprzedniego punktu;
   (5) analogicznie stworzyć klasę `ZielonaCzcionkaDekorator` dziedziczącą po `DekoratorNapisu` ;
   (6) wypróbować kod poprzez wykonanie:

```
Napis a = new ZielonaCzcionkaDekorator(new CzerwoneTloDekorator(new
     KrotkiNapis(„Testowy napis"))) ;
a.wypisz();
```

   (7) stworzyć analogiczną do poprzedniej klasę `DlugiNapis`, która będzie wypisywała dzieląc go na kolejne linie jeżeli będzie dłuższy niż 20 znaków.

6. Pozostałe kody modyfikujące tekst:

```
public static final String ANSI_RESET = "\u001B[0m";
public static final String ANSI_BLACK = "\u001B[30m";
public static final String ANSI_RED = "\u001B[31m";
public static final String ANSI_GREEN = "\u001B[32m";
public static final String ANSI_YELLOW = "\u001B[33m";
public static final String ANSI_BLUE = "\u001B[34m";
public static final String ANSI_PURPLE = "\u001B[35m";
public static final String ANSI_CYAN = "\u001B[36m";
public static final String ANSI_WHITE = "\u001B[37m";
public static final String ANSI_BLACK_BACKGROUND = "\u001B[40m";
public static final String ANSI_RED_BACKGROUND = "\u001B[41m";
public static final String ANSI_GREEN_BACKGROUND = "\u001B[42m";
public static final String ANSI_YELLOW_BACKGROUND = "\u001B[43m";
public static final String ANSI_BLUE_BACKGROUND = "\u001B[44m";
public static final String ANSI_PURPLE_BACKGROUND = "\u001B[45m";
public static final String ANSI_CYAN_BACKGROUND = "\u001B[46m";
public static final String ANSI_WHITE_BACKGROUND = "\u001B[47m";
```