Programowanie Zespołowe

Dobre Praktyki

dr Rafał Skinderowicz
mgr inż. Michał Maliszewski

„Parafrazując klasyka: Jeśli piszesz w Javie – pisz w Javie”

- Rafał Ciepiela
Principal Software Developer
Cadence Design Systems

Czysty Kod

- Ułatwia utrzymanie i poprawianie błędów
- Ułatwia przepływ wiedzy między zespołem programistów
- Nie posiada długu technicznego
- Jest wspierany przez testy jednostkowe i automatyczne
- Posiada aktualną dokumentację
- Nie zawiera duplikatów
- Jasno wyraża intencje programisty
- Zawiera minimalną ilość klas i metod
- Przestrzega ogólnie przyjętych zasad i reguł

Jakość kodu

Jakość kodu podnoszą:

- Wzorce projektowe (OOP)
- Jasna i prosta architektura systemu
- DoD (Definition of Done)
- Code Review
- Testy jednostkowe
- Agile Software Development
- Narzędzia do sprawdzania jakości kodu (Sonar, PMD, FindBugs itp.)

Code Review

Czynność polegająca na sprawdzeniu kodu (jego formy, wyglądu i sposobu rozwiązania problemu) przez kogoś poza twórcą.

- Jeden z najważniejszych elementów metodyk wytwarzania oprogramowania
- Podstawa programowania ekstremalnego
- Jedno z głównych założeń Feature Driven Development
- Wykonywana przed włączeniem kodu do systemu kontroli wersji
- Wykonywana przed przekazaniem kodu do testowania

Programowanie parami

- Jedną z form Code Review
- Pierwotnie część XP (eXtreme Programming)
- Programiści piszą w parach: jedna osoba pisze kod, druga obserwuje pierwszą, zgłasza poprawki i zadaje pytania
- Programiści zamieniają się rolami
- Wzajemna nauka
- Przepływ wiedzy
- Zwiększona jakość kodu

TDD

Test-driven development jest techniką tworzenia oprogramowania z grupy metodyk zwinnych. Programowanie techniką TDD wyróżnia się tym, że najpierw pisane są testy jednostkowe funkcjonalności, która jeszcze nie istnieje. Wielokrotne powtarzane są kroki:

- Napisz test jednostkowy sprawdzający dodawaną funkcjonalność (test w tym momencie nie powinien się udać)
- Zaimplementuj funkcjonalność (test powinien się udać)
- Zrefaktoryzuj napisany kod do ogólnie przyjętych standardów (OOP, DoD itd.)

„Stosunek czasu spędzonego na czytaniu kodu w porównaniu do czasu pisania wynosi ponad 10 do 1, dlatego tworzenie czytelnego kodu ułatwia pisanie”

Robert C. Martin, Czysty Kod

Czytelność kodu

```
public static boolean func(final int n1, final int n2, final int n3) {
    if(n2 > n1) {
        if(n3 > n2) {
            return true;
        } else {
            return false;
        }
    } else if(n2 == n1) {
        if(n3 == n1) {
            return true;
        } else {
            return false;
        }
    } else {
        return false;
    }
}
```

„Pierwszą zasadą każdej funkcji jest, że powinna być mała”

Robert C. Martin, Czysty Kod

Czytelność kodu

```
public static boolean isInOrder(final int n1, final int n2, final int n3) {
    return n1 <= n2 && n2 <= n3;
}
```

SOLID

Za jedne z podstawowych zasad zachowania czystości w kodzie zorientowanym obiektowo uchodzą zasady SOLID, zaproponowane przez Roberta C. Martina. Mnemonik można rozpisać jako:

- **S** – Single responsibility principle (zasada jednej odpowiedzialności) – klasa powinna mieć tylko jedną odpowiedzialność (nigdy nie powinien istnieć więcej niż jeden powód do modyfikacji klasy)
- **O** – Open/closed principle (zasad otwarte-zamknięte) – Klasa powinna być otwarta na rozszerzenie, ale zamknięta na modyfikacje

SOLID c.d.

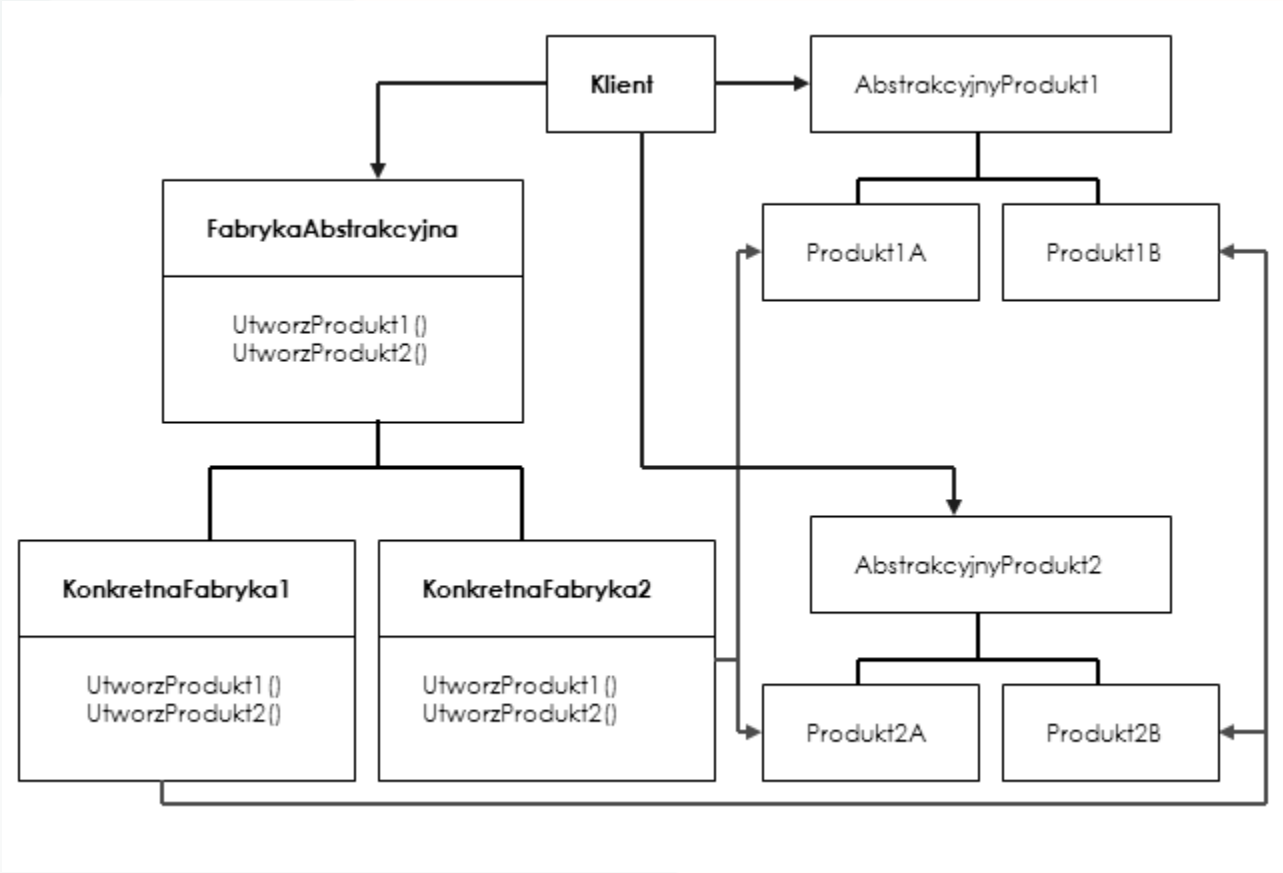
- **L** – Liskov substitution principle (zasada podstawienia Liskov) – korzystanie z funkcji klasy bazowej musi być możliwe również w przypadku podstawienia instancji klas pochodnych
- **I** – Interface segregation principle (zasada segregacji interfejsów) – Klienci nie powinni zależeć od interfejsów, których nie używają
- **D** – Dependency inversion principle (zasada odwrócenia zależności) – wysokopoziomowe moduły nie powinny zależeć od modułów niskopoziomowych – zależności między nimi powinny wynikać z abstrakcji

Wzorce projektowe

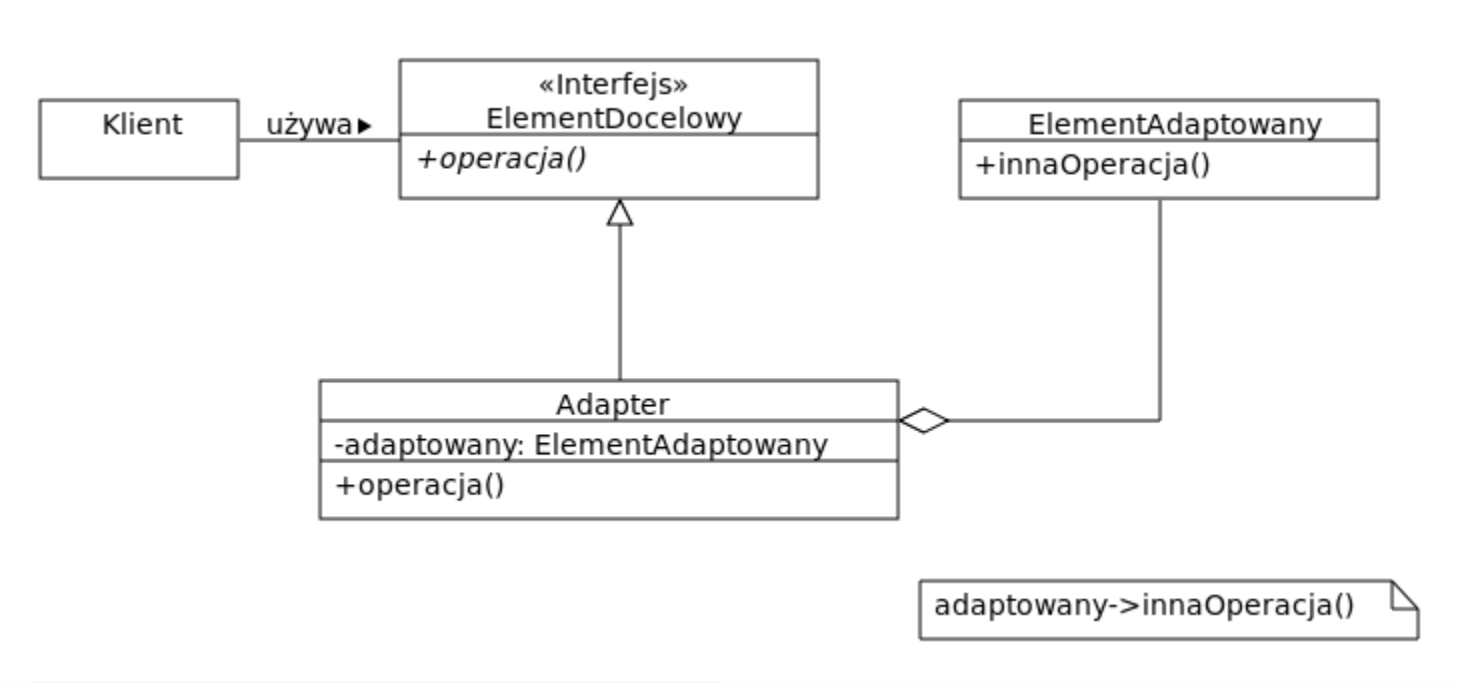
Uniwersalne, sprawdzone w praktyce rozwiązania często pojawiających się problemów z zakresu inżynierii oprogramowania (OOP). Wyróżniamy 3 rodzaje wzorców:

- **Kreacyjne** (fabryka abstrakcyjna, singleton)
- **Strukturalne** (adapter, dekorator, most, fasada)
- **Czynnościowe** (obserwator, strategia, stan, metoda szablonowa)

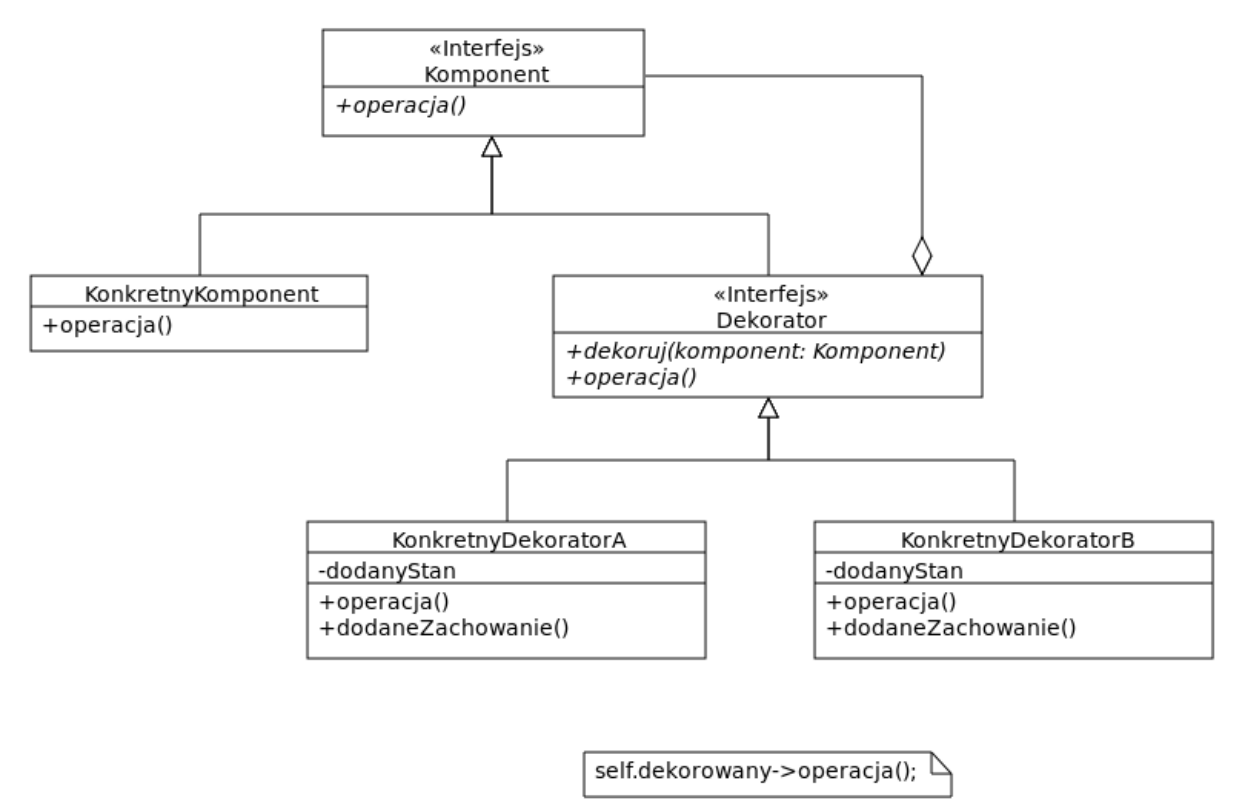
Fabryka Abstrakcyjna



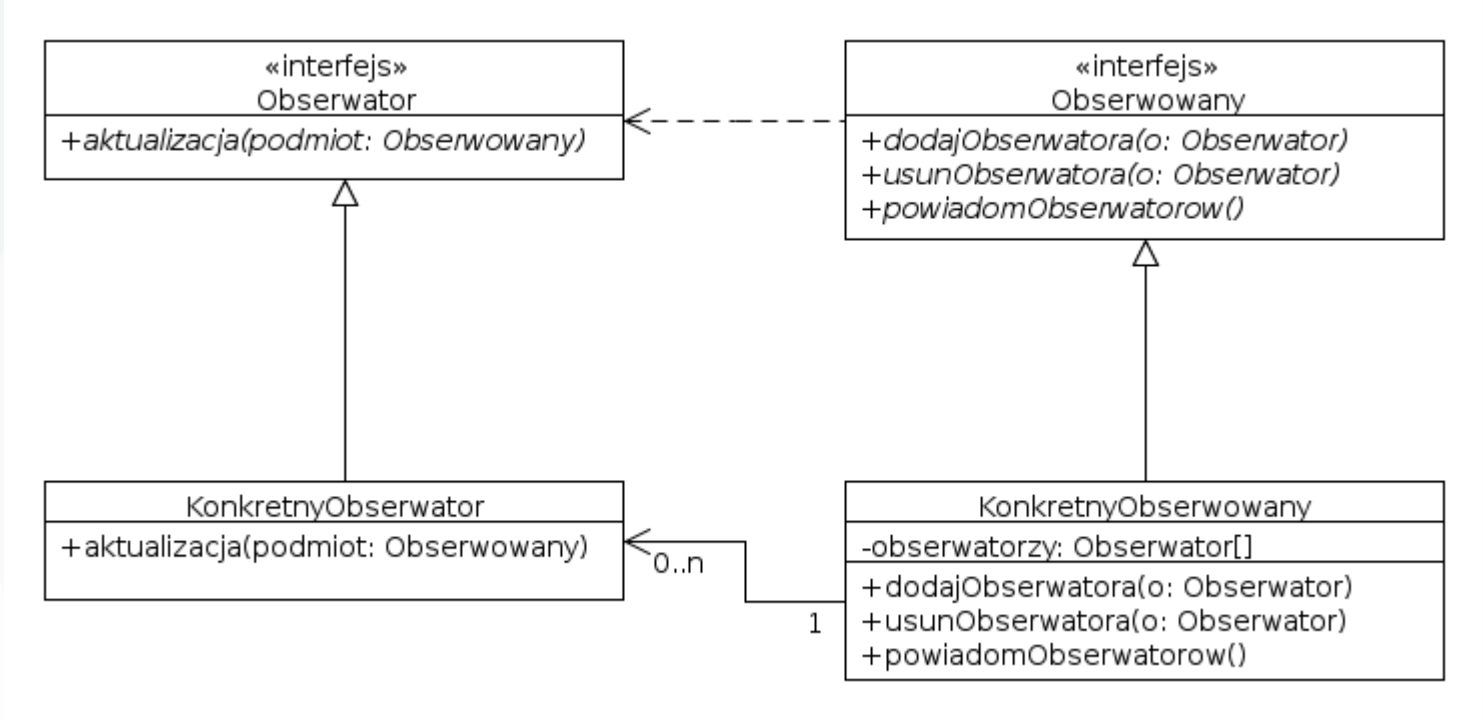
Adapter


Klient
używa
«Interfejs»
ElementDocelowy
+operacja()
ElementAdaptowany
+innaOperacja()
Adapter
-adaptowany: ElementAdaptowany
+operacja()
adaptowany->innaOperacja()

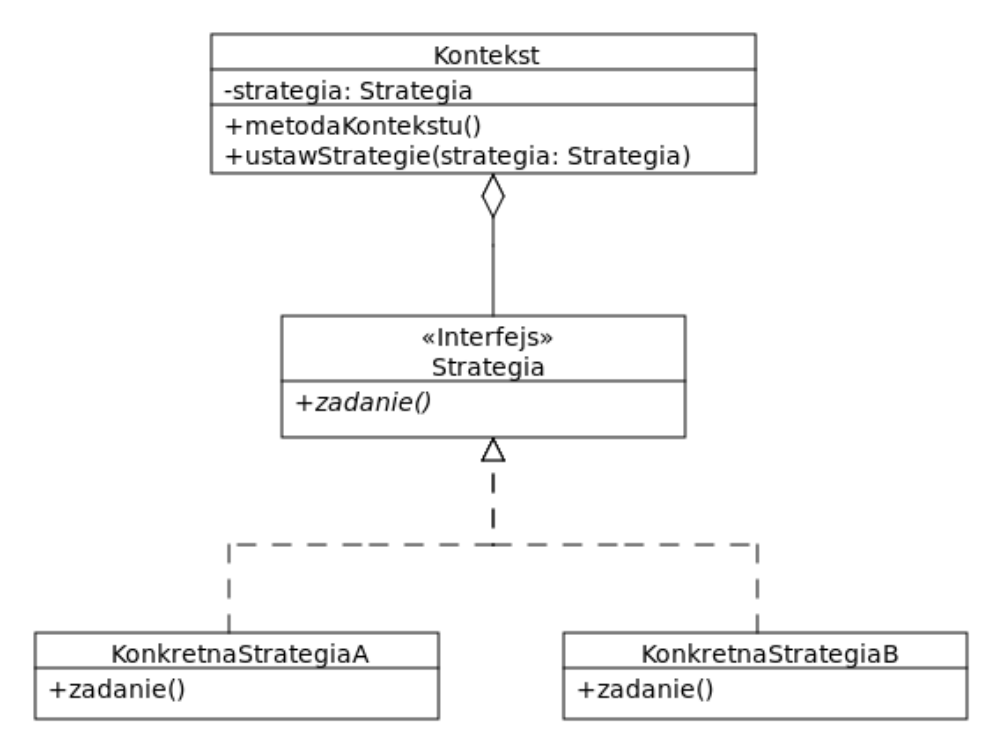
Dekorator


«Interfejs»
Komponent
+operacja()
KonkretnyKomponent
+operacja()
«Interfejs»
Dekorator
+dekoruj(komponent: Komponent)
+operacja()
KonkretnyDekoratorA
-dodanyStan
+operacja()
+dodaneZachowanie()
KonkretnyDekoratorB
-dodanyStan
+operacja()
+dodaneZachowanie()
self.dekorowany->operacja();

Obserwator


«interfejs»
Obserwator
+aktualizacja(podmiot: Obserwowany)
«interfejs»
Obserwowany
+dodajObserwatora(o: Obserwator)
+usunObserwatora(o: Obserwator)
+powiadomObserwatorow()
KonkretnyObserwator
+aktualizacja(podmiot: Obserwowany)
KonkretnyObserwowany
-obserwatorzy: Obserwator[]
+dodajObserwatora(o: Obserwator)
+usunObserwatora(o: Obserwator)
+powiadomObserwatorow()
0..n
1

Strategia


Kontekst
-strategia: Strategia
+metodaKontekstu()
+ustawStrategie(strategia: Strategia)
«Interfejs»
Strategia
+zadanie()
KonkretnaStrategiaA
+zadanie()
KonkretnaStrategiaB
+zadanie()

Ćwiczenie 1

```
public class Customer {

    private BigInteger id;
    private String name;

    public BigInteger getId() {
        return id;
    }

    public void setId(final BigInteger id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(final String string) {
        name = string;
    }

    public String toXML() {
        final StringBuilder xml = new StringBuilder();
        xml.append("<Customer>");
        xml.append("<Id>").append(name).append("</Id>");
        xml.append("<Name>").append(name).append("</Name>");
        xml.append("</Customer>");
        return xml.toString();
    }
}
```

Ćwiczenie 2

```
public void printAccountDetails(final Customer account) {
    // print summary
    System.out.println(MessageFormat.format("Account Id: {0}", account.getId()));
    System.out.println(MessageFormat.format("Account Name: {0}", account.getName()));

    // print history
    for(final Transaction tx : account.getTransactions()) {
        System.out.println(tx.toString());
    }
}
```

Ćwiczenie 3

Pracując w parach napiszcie program, który wyświetli zadany tekst w postaci kodu Morse'a. Język programowania oraz sposób implementacji jest dowolny.

- Pierwsza osoba pisze kod przez 3 minuty
- Druga osoba przygląda się tworzeniu, zgłasza poprawki i zadaje pytania
- Następuje zamiana ról.

Dziękuję za uwagę