laboratoria 9—10

Modelowanie interakcji między klasami i obiektami

Cel zajęć: Budowa modelu interakcji klas i obiektów oprogramowania w postaci diagramów sekwencji.
Zastosowanie wybranych wzorców projektowych zgodnie z zasadami SOLID.
Implementacja struktury i zachowania tworzonego oprogramowania na podstawie jego modelu.

Źródła wiedzy: Wykład IO: *Języki graficznego modelowania* (dostępny też w *ePortalu Pwr*).
Wykład IO: *Modelowanie zachowania diagramami interakcji* (dostępny też w *ePortalu PWr*).
Przykład: *Miniprojekt Biblioteka* (dostępny też w *ePortalu PWr*).
Tutorial *Visual Paradigm*: *How to Draw Sequence Diagram?*

Zawartość instrukcji: 2 zadania – ich wykonanie jest oceniane.
1 ćwiczenie pomocnicze.
Spis rzeczy, które należy umieścić w sprawozdaniu z wykonania zadań.
Spis błędów i braków mogących obniżyć ocenę.

Zadania

Zadanie 1

Budowa modelu interakcji klas i obiektów oprogramowania w postaci diagramów sekwencji.

Należy utworzyć diagramy sekwencji w *Visual Paradigm* przedstawiające interakcje klas i obiektów w wykonywaniu operacji realizujących przypadki użycia wybrane w poprzednich etapach laboratoriów. Należy przy tym stosować założenia SOLID i odpowiednie wzorce projektowe określone w poprzednim etapie laboratoriów.

Diagram sekwencji powinien modelować operację klasy z wykonanego modelu klas i przedstawiać interakcje zachodzące przede wszystkim między klasami i obiektami z wykonanego modelu klas. Te interakcje powinny być operacjami bezpośrednio wywoływanymi z ciała tej jednej operacji.

Jeden diagram sekwencji powinien modelować operację inicjującą realizację przypadku użycia; następny diagram sekwencji – operację przez nią używaną itd. rekurencyjnie, póki w ciele danej operacji jest interakcja między klasami i obiektami z wykonanego modelu klas (z pominięciem symulacji działania warstwy prezentacji).

Operacje użyte na diagramach sekwencji powinny być też umieszczone w odpowiednich klasach na diagramach klas.

Na diagramie sekwencji, modelującym interakcje dla danej operacji, należy umieścić m.in.:

– w pełni nazwane linie życia uczestników tych interakcji, czyli inicjatorów i wykonawców tej operacji i operacji zawartych w jej ciele;

– w pełni nazwane i ponumerowane hierarchicznie (1, 1.1 itd.) wiadomości synchroniczne, zwrotne i tworzące obiekt, czyli modelujące wywołanie tej operacji i operacji zawartych w jej ciele;

– odpowiednie do algorytmu interakcji połączone fragmenty (musi być przykład użycia *loop* i *opt* lub *alt*).

Przy każdym diagramie należy umieścić skrót kodu operacji (jej nagłówek i ciało) modelowanej przez wiadomość nr 1 tego diagramu, zgodny z zawartością diagramu. Komentarze i fragmenty ciała operacji, niepokazane na diagramie i niezwiązane z modelowanymi tam interakcjami, można pominąć w skrócie kodu.

Linie życia w zadaniu 1

Diagram sekwencji modeluje interakcje zachodzące między jego uczestnikami – klasami lub obiektami – na osiach czasu ich aktywności, czyli *liniach życia.* Ze względu na operację modelowaną przez cały diagram, linią życia może być:

– *: NazwaKlasy,* czyli instancja klasy, która wykonuje tę operację (w nazwie linii życia nie ma nazwy instancji, gdyż jest nieznana dla ciała tej operacji);

– *nazwaObiektu : NazwaKlasy,* czyli instancja klasy użyta w ciele tej operacji;

– *NazwaKlasy : NazwaKlasy,* czyli klasa, której statyczne operacje wykonywane są w ciele tej operacji.

Tą klasą może być dowolna klasa zawarta na diagramie klas lub dowolna klasa ze standardowych pakietów języka programowania, np. *java.util.ArrayList.*

Interakcje, czyli wiadomości w zadaniu 1

Diagram sekwencji modeluje interakcje zachodzące między jego uczestnikami – klasami lub obiektami, zwanymi *liniami życia* – podczas wykonywania jednej operacji zaimplementowanej w jednej (zwykle pierwszej od lewej) z tych linii życia (w klasie). Interakcją jest przekazanie sterowania (i danych, jeśli trzeba), czyli zainicjowanie przez jedną linię życia operacji wykonywanej przez inną (lub niekiedy tę samą) linię życia, czyli żądanie wykonania tej operacji.

Pojedynczą interakcję modeluje przesłanie wiadomości od nadawcy (linii życia inicjującej operację) do odbiorcy (linii życia wykonującej operację). Zależnie od operacji są to zwykle:

– wiadomość *tworząca obiekt* – konstruktor instancji klasy – z opisem *NazwaKlasy(nazwaParametru : TypParametru,…)*
– wiadomość *synchroniczna* wywołania operacji (*call message*) – inna operacja klasy – z opisem
nazwaOperacji(nazwaParametru : TypParametru,…) : TypWyniku
może ona być:
  • zwykła, jeśli nadawcą i odbiorcą są różne linie życia;
  • *zgubiona*, jeśli żadna z linii życia tego diagramu nie jest jej odbiorcą (odbiorca jest tu niezdefiniowany);
  • *znaleziona*, jeśli żadna z linii życia tego diagramu nie jest jej nadawcą (nadawca jest tu niezdefiniowany);
  • *własna*, jeśli nadawcą i odbiorcą jest ta sama linia życia (w ciele operacji klasy wykonywana jest jej inna operacja) z opisem *nazwaOperacji(nazwaParametru : TypParametru,…) : TypWyniku : zmiennaOtrzymującaWynik* jeśli operacja coś zwraca, a *zmiennaOtrzymującaWynik* jest potem wykorzystana na diagramie;
  • *rekursywna*, jeśli nadawcą i odbiorcą jest ta sama linia życia i powoduje ona wysłanie innej wiadomości;
– wiadomość *zwrotna*, towarzysząca jakiejś wiadomości synchronicznej, czyli powrót z wykonania operacji klasy,
z opisem *nazwaOperacji(wartośćParametru,…) : wynikLubJegoTypJeśliZGóryNieJestZnany*
lub *zmiennaOtrzymującaWynik = nazwaOperacji(wartośćParametru,…) : wynikLubJegoTypJeśliZGóryNieJestZnany*
jeśli *zmiennaOtrzymującaWynik* jest potem wykorzystana na diagramie.

Odbiorcą wiadomości zwrotnej jest nadawca związanej z nią wiadomości synchronicznej. Jeśli ta wiadomość synchroniczna jest wiadomością znalezioną, to wiadomość zwrotna jest wiadomością zgubioną. Wiadomość zwrotną można pominąć, jeśli operacja niczego nie zwraca (jest typu *void*). Wiadomość zwrotną trzeba pominąć, jeśli operację modeluje wiadomość własna.

Zagnieżdżoną operację modeluje sekwencja (od góry do dołu) wiadomości odpowiadających tym operacjom w kolejności od zewnętrznej do wewnętrznej. Przykładowo, dla operacji *this.op1(coś.op2().op3(), this.op4())* obiektu *obiekt* są to kolejno:

1) wiadomość własna lub rekurencyjna linii życia *obiekt: op1()*,
2) wiadomość z linii życia *obiekt* do linii życia *coś*: *op2()*,
3) wiadomość z linii życia *obiekt* do linii życia *coś*: *op3()*,
4) wiadomość własna lub rekurencyjna linii życia *obiekt: op4()*.

Połączone fragmenty w zadaniu 1

Połączony fragment (ramka) otacza część diagramu, nadając otoczonym interakcjom znaczenie określone przez jego typ, czyli *operator*. Może mieć 1 lub więcej części, czyli *operandów*, które mogą mieć swój warunek wykonania, czyli *dozór*. Najczęściej są to:

– *opcja* (operator *opt*; 1 operand) – instrukcja warunku z tylko 1 *if* i brakiem *else*;
– *alternatywa* (operator *alt*; 2 lub więcej operandów) – instrukcja wyboru z *if* z dowolną liczbą *else if* lub *else* albo instrukcja wyboru *switch* z 2 lub więcej *case;*
– *pętla* (operator *loop*; 1 operand; określenie liczby wykonań pętli) – instrukcja pętli *for*, *while* lub *do...while*.

Inne rodzaje połączonych fragmentów też się mogą przydać, zależnie od algorytmu modelowanej operacji.

Połączony fragment *loop* nie jest przystosowany do pętli *for each*, ale można to osiągnąć na kilka sposobów:

1) Zamiast liczby wykonań pętli podaje się *(nazwaKolekcji.size()).*
2) Zamiast dozoru podaje się *[elementKolekcji : nazwaKolekcji].*
3) Zamiast dozoru podaje się *[for each nazwaKolekcji item].*
4) Tworzy się i używa iteratora kolekcji, a w dozorze podaje się *[iterator.hasNext()]*;
5) Tworzy się specjalny stereotyp (np. «forEach») połączonego fragmentu *loop* – zalecane w zadaniu 1.
6) Tworzy się specjalny profil (rodzaj) połączonego fragmentu *loop*.

Jeśli w dozorze operandu (czyli w nawiasach *[…])* jest wywołanie operacji klasy lub obiektu, to wiadomość z tą operacją powinna być umieszczona w tym operandzie na samym początku zawartej w nim sekwencji wiadomości.

Dekompozycja interakcji w osobne diagramy sekwencji:

Najpierw należy wykonać diagram sekwencji dla operacji inicjującej realizację wybranego przypadku użycia przez klasę fasadową komponentu kontroli. Ta operacja będzie wiadomością nr 1 typu *wiadomość znaleziona*, a operacje wywoływane w ciele tej operacji (wykonywane przez tę samą lub inną klasę) będą wiadomościami nr 1.1, 1.2 itd. wychodzącymi

z aktywacji (pionowy prostokąt na linii życia) wiadomości nr 1 do linii życia odpowiednich uczestników interakcji, czyli klas lub obiektów wykonujących te operacje.

Zaleca się, aby diagram sekwencji modelował tylko te interakcje, które są bezpośrednio zawarte w ciele operacji wiadomości nr 1. Jeśli wiadomość 1.1, 1.2 itd. jest wywołaniem operacji innego obiektu lub klasy z wykonanego modelu klas, to zaleca się, aby z aktywacji wiadomości 1.1, 1.2 itd. nie wychodziły kolejne wiadomości na tym samym diagramie. Interakcje zawarte w ciele operacji z takiej wiadomości (np. 1.1) można przedstawić na osobnym diagramie sekwencji, podobnie jak operację nr 1. Tę zasadę stosuje się rekurencyjnie aż do przyjętego poziomu szczegółowości. Dotyczy to też rekurencyjnego wykonania operacji w jej własnym ciele. Można odstąpić od tej zasady, jeśli taka operacja (np. 1.1) jest wykonywana tylko jeden raz, tylko w tej jednej operacji (nr 1), tylko w tej jednej klasie.

Przykładowo, jeśli diagram modeluje wykonanie operacji *A()*, której ciało zawiera wykonanie operacji *B()*, to z aktywacji rozpoczętej otrzymaniem wiadomości *A()* wychodzi wiadomość *B()* do odpowiedniej linii życia, na której rozpoczyna swoją aktywację. Z tej drugiej aktywacji, czyli rozpoczętej przez wiadomość *B(),* nie powinny wychodzić wiadomości, jeśli:

– operacja *B()* też zawiera interakcje między obiektami i klasami z wykonanego modelu klas;
– operacja *B()* jest wykonywana więcej niż 1 raz na tym samym lub różnych diagramach;
– operacja *B()* nazywa się *A()*, czyli jest jej rekurencyjnym wykonaniem.

Wówczas osobny diagram powinien modelować wykonanie operacji B().

Zadanie 2

Implementacja struktury i zachowania tworzonego oprogramowania na podstawie jego modelu.

Jeśli modele klas, wykonane w poprzednim etapie laboratoriów, zostały zmienione (dodano, usunięto lub zmieniono klasę, jej atrybut, operację lub relację), to należy ponownie wykonać kod wszystkich klas na podstawie ich modeli.

Kod klasy powinien zawierać deklarację klasy oraz jej atrybutów i operacji, uwzględniając związki tej klasy z nią samą i z innymi klasami (uogólnienie, realizacja, asocjacja itd.).

Należy uzupełnić kod klas tak, aby możliwe było testowe działanie całego oprogramowania w trybie tekstowym w celu realizacji przypadków użycia wybranych w poprzednich etapach laboratoriów. Należy więc zaimplementować:

– operacje konieczne do uruchomienia całego oprogramowania, zaczynając od głównej operacji (*main*);
– operacje odpowiedzialne za realizację wybranych przypadków użycia, wykorzystywane przez nie operacje (itd.) – zgodnie z diagramami sekwencji z zadania 1.

Idea testowego działania wykonywanego oprogramowania

Uruchomienie oprogramowania zaczyna się od głównej operacji (*main*) będącej w warstwie kontroli, która kolejno:

1) Przygotowuje dane testowe – tworzy obiekty z testowymi wartościami wejściowymi dla operacji przypadków użycia.
2) Przygotowuje klasy i obiekty tworzące poszczególne części oprogramowania – tworzy i inicjuje ich instancje, łączy je ze sobą, ustawia je w stan początkowy itp.
3) Rozpoczyna testowe wykonanie operacji wybranego przypadku użycia (będącego w relacji asocjacji z inicjującym go aktorem) – wywołuje ją w fasadowej klasie warstwy kontroli.
4) W pełni realizuje ten przypadek użycia przy pomocy tej operacji, która w tym celu wywołuje operacje tej lub innych klas itd. zgodnie z diagramami czynności modelującymi realizację tego przypadku użycia i użytymi wzorcami projektowymi.
5) Wyświetla wynik tej operacji – w oknie terminala w trybie tekstowym.
6) Rozpoczyna testowe wykonanie operacji kolejnego wybranego przypadku użycia (jak wyżej) itd.

Wykonanie kodu klas:

Najpierw należy automatycznie wykonać kod klas w *Visual Paradigm* na podstawie diagramów klas. Zaleca się użycie okna *Instant Generator* (menu *Tools* → *Code* → *Instant Generator)* i w nim m.in.:

– wybrać język programowania,
– wybrać katalog docelowy (*Output directory*) na tworzone pliki z kodem,
– zaznaczyć diagramy klas, na podstawie których ma powstać kod.

Następnie należy w wybranym środowisku IDE ręcznie poprawić i uzupełnić kod klas o wyjaśniające go komentarze i dodatkowy kod, w tym ciało operacji tak, aby możliwe było uruchomienie i przetestowanie oprogramowania zgodnie z ideą opisaną wyżej. Ciało operacji przedstawionej na diagramie sekwencji powinno być z nim zgodne. Diagram sekwencji pokazuje tylko interakcję jego uczestników. Kod operacji zwykle ma też dodatkowe rzeczy (w tym komentarze i operacje), których nie ma na diagramie, ale które są konieczne do zrozumienia kodu lub do prawidłowego działania oprogramowania.

Zadanie 2 opracowano częściowo na podstawie instrukcji 6 i instrukcji 7 „Laboratorium z przedmiotu: Inżynieria Oprogramowania W04ITE-SI0011G”, autorstwa dr inż. Zofii Kruczkiewicz.

Ćwiczenia

Ćwiczenie 1
Model interakcji na podstawie kodu operacji.

Proszę narysować diagram sekwencji modelujący operację *FabrykaKsiążkiZgubionej.utwórzKsiążkę(String opisCsv)*, z *Miniprojektu Biblioteka*, zgodnie z poniższym kodem napisanym w języku *Java*:

```
public class FabrykaKsiążkiZgubionej {
    private IKsiążka książka;
    private Czytelnik czytelnik;

    // ... ukryto  konstruktor klasy i inne jej operacje

    public IKsiążka utwórzKsiążkę(String opisCsv) {
       CSV csv = new CSV();
       IKsiążka nowaKsiążka = null;
       if (csv.czyData(opisCsv) && this.książka != null) {
                nowaKsiążka = new ZgubionaKsiążka(this.książka, opisCsv);
       }
       else if (!csv.czyData(opisCsv) && !csv.znajdźDatęWypożyczeniaKsiążki(opisCsv).isEmpty() && this.czytelnik != null) {
                FabrykaKsiążkiWypożyczonej fabryka = new FabrykaKsiążkiWypożyczonej(this.książka, this.czytelnik);
                nowaKsiążka = new ZgubionaKsiążka(fabryka.utwórzKsiążkę(opisCsv), csv.znajdźDatęZgubieniaKsiążki(opisCsv));
       }
       else if (!csv.czyData(opisCsv) && csv.znajdźDatęWypożyczeniaKsiążki(opisCsv).isEmpty()) {
                FabrykaKsiążki fabryka = new FabrykaKsiążki();
                nowaKsiążka = new ZgubionaKsiążka(fabryka.utwórzKsiążkę(opisCsv), csv.znajdźDatęZgubieniaKsiążki(opisCsv));
       }
       return nowaKsiążka;
    }
}
```

Do wykonania ćwiczenia przydadzą się wiadomości: zwykła synchroniczna, znaleziona synchroniczna, zwykła zwrotna, zgubiona zwrotna i tworząca obiekt. Operacje, które należy uwzględnić, są podkreślone.

Sprawozdanie

Co powinno być na początku sprawozdania:

– Autorzy (imię, nazwisko, nr albumu).
– Nazwa tworzonego oprogramowania (zwięzły opis jego dziedziny).
– Temat tego etapu laboratoriów (*Modelowanie interakcji między klasami i obiektami*).

Co powinno być w treści sprawozdania:

– Zaktualizowane diagramy klas wykonane w poprzednim etapie laboratoriów.
– Diagramy sekwencji wykonane w zadaniu 1.
– Przy każdym diagramie sekwencji – skrót kodu operacji modelowanej przez ten diagram.
– Pełny kod klas wykonany w zadaniu 2 (tekst, a nie obraz ze zdjęciem ekranu), zawierający:
  • klasy w kolejności ich w pełni kwalifikowanych nazw (czyli wraz ze ścieżką pakietów, np. *pakiet.klasa.java*),
  • nazwę pliku z definicją klasy (nad tą definicją),
  • całą zawartość tego pliku,
  • numerację linii,
  • komentarze do mało oczywistych rzeczy,
  • wcięcia i inne formatowania ułatwiające czytelność.

Błędy i braki

Jeśli opisane niżej rzeczy mają miejsce, ocena za ten etap laboratoriów może być niższa:

- Nie wykonano dekompozycji interakcji w osobne diagramy sekwencji.
- Jeden diagram sekwencji modeluje dwie lub więcej osobnych operacji, czyli niezależnych sekwencji wiadomości.
- Jedną z linii życia jest aktor.
- Nazwa obiektu, klasy, operacji, parametru operacji lub zmiennej jest inna niż w kodzie i na diagramie klas.
- Typ parametru lub wyniku operacji w nazwie wiadomości jest inny niż na diagramie klas.
- W nazwie linii życia brak nazwy klasy poprzedzonej dwukropkiem lub brak dwukropka.
- W nazwie linii życia obiektu brak nazwy obiektu przed dwukropkiem, chociaż ta nazwa jest w kodzie.
- W nazwie linii życia obiektu jest nazwa obiektu przed dwukropkiem, chociaż żadnej nazwy nie ma w kodzie.
- Nazwa linii życia klasy wykonującej operacje statyczne jest inna niż *NazwaKlasy : NazwaKlasy*.
- Brak całości lub części opisu wiadomości.
- Brak hierarchicznego numerowania wiadomości.
- Brak przykładu zastosowania wiadomości: synchronicznej, zwrotnej lub tworzącej obiekt.
- Wywołanie operacji (nie konstruktora) jest wiadomością asynchroniczną zamiast synchroniczną.
- Sekwencje na diagramie zaczynają się inaczej niż od synchronicznej wiadomości zgubionej dla operacji nr 1 modelowanej tym diagramem.
- Wiadomość tworząca obiekt wskazuje na oś linii życia zamiast na jej nagłówek.
- Wiadomość kończąca operację jest innego typu niż wiadomość zwrotna.
- Brak wiadomości zwrotnej kończącej operację coś zwracającą.
- Jest wiadomość zwrotna kończąca operację konstruktora.
- Wiadomość zwrotna kończąca operację trafia do innej linii życia niż ta, z której wyszła wiadomość synchroniczna inicjująca tę operację.
- Brak całości lub części opisu wiadomości zwrotnej, chociaż jest w kodzie.
- Przed nazwą operacji w opisie wiadomości jest nazwa wykonującego ją obiektu, np. *this.operacja()* lub *coś.operacja()*.
- Brak przykładu zastosowania połączonego fragmentu *loop* lub połączonego fragmentu *opt* lub *alt*.
- Połączony fragment nie obejmuje wszystkich linii życia, których dotyczą zawarte w nim wiadomości.
- Połączony fragment *opt* ma więcej niż jeden operand (część).
- Połączony fragment *alt* ma tylko jeden operand (część).
- Operand połączonego fragmentu *opt* lub *alt* nie ma dozoru, lub jego dozór jest niezgodny z kodem.
- Operand połączonego fragmentu *loop* ma inny opis (liczbę wykonania pętli lub dozór) niż w kodzie.
- Na początku sekwencji wiadomości w operandzie brak wiadomości dla operacji zawartej w dozorze tego operandu.
- Połączony fragment *ref* nie obejmuje wszystkich linii życia, które są też na diagramie, do którego jest referencją.
- Połączony fragment *ref* ma inną nazwę niż diagram, do którego jest referencją.
- Połączony fragment ref zawiera wysłanie lub odebranie jakiejś wiadomości.
- Aktywacja (pionowy prostokąt na osi linii życia) zaczyna się wcześniej niż tam, gdzie wchodzi wiadomość inicjująca wykonanie operacji modelowane tą aktywacją.
- Aktywacja nie kończy się tam, gdzie operacja, której wykonanie modeluje, ale łączy się z wykonaniem kolejnej operacji przez tę linią życia.
- Naruszono bez praktycznego powodu zasady SOLID, np.:
  - klasa ma więcej niż jedną odpowiedzialność;
  - dynamiczne (czyli w czasie działania programu) rozbudowywanie klasy odbywa się przy pomocy dziedziczenia klas, zamiast ich kompozycji/agregacji.
  - klasa dynamicznie zależna od różnych klas (zmiana zależności w czasie działania programu) jest z nimi związana sztywno, zamiast przy pomocy ich abstrakcji (klasy abstrakcyjnej lub interfejsu);
  - podklasa (w relacji realizacji lub uogólnienia) wywołuje operacje swojej nadklasy (naruszenie tzw. reguły *Hollywood*);
- Brak kodu lub fragmentu kodu klasy lub operacji obecnej na diagramie klas lub na diagramie sekwencji.
- Kod oprogramowania przeczy diagramowi klas lub diagramowi sekwencji.
- Kod klas, w szczególności kod operacji głównej (*main*), uniemożliwia przewidziane treścią zadania testowe uruchomienie i działanie oprogramowania w zakresie realizacji wybranych przypadków użycia.
- Kod nie jest sformatowany lub jest niepraktycznie sformatowany, np. brak komentarzy, wcięć czy numeracji linii.

Ta lista zawiera typowo pojawiające się błędy i braki i nie wyczerpuje powodów, dla których ocena za laboratoria może być niższa.