

Języki i paradygmaty programowania:

Laboratorium nr 11

Podstawowe paradygmaty programowania
obiektowego - wprowadzenie. Java - Obsługa
wyjątków. Kolekcje.

2017-2018

mgr inż. Przemysław Walkowiak

dr inż. Michał Ciesielczyk

Instrukcja

W czasie pisania programu pamiętaj o:

1. dbaniu o czytelność kodu (odpowiednie formatowanie kodu, nazewnictwo zmiennych adekwatne do ich znaczenia, komentarze),
2. dbaniu o czytelność interfejsu z użytkownikiem (w sposób jawny pytaj użytkownika jakie dane ma podać oraz opisz wyniki, które zwracasz),
3. przed fragmentem implementującym poszczególne zadania umieść komentarz: `/*Zadanie X */` oraz wypisz na ekranie analogiczny komunikat (X jest numerem zadania),
4. każde zadanie umieść w oddzielnej klasie z odpowiednimi metodami,
5. zaimplementuj menu wyboru zadania, a następnie wykorzystując pętle **do-while** oraz konstrukcję **switch** wykonaj odpowiedni fragment kodu,
6. w zadaniach wymagających udzielenia komentarza bądź odpowiedzi, należy umieścić go w kodzie programu (np. w postaci komentarza albo wydrukować na ekranie),
7. w zadaniach polegających na zaprojektowaniu klasy należy utworzyć jej instancję i wykorzystać zaimplementowaną funkcjonalność.

Wprowadzenie

Obsługa wyjątków

Mechanizm obsługi wyjątków w Javie, podobnie jak w C++, pozwala reagować na wystąpieniu zdarzeń (w szczególności błędów) zmieniających prawidłowy przebieg wykonywania się programu. Instrukcje, których wykonanie może skutkować wywołaniem wyjątku, umieszczane są w bloku **try** (linia 2). Obsługa wyjątków z bloku **try** wykonywana jest w następujących zaraz po nim blokach **catch**. Dla każdego bloku **try** może przypadać jeden lub więcej bloków **catch** – w zależności od liczby możliwych typów wyjątków do obsługi.

Przykładowo, blok **catch**:

- w linii 3 pozwala na obsługę kilku typów wyjątków jednocześnie;
- w linii 6 pozwala na obsługę wyjątku `re` będącego instancją klasy `RuntimeException`;
- w linii 9 przechwytywałby każdy wyjątek nieobsłużony w żadnym z poprzednich bloków.

```
try {  
    /* do something that may fail */  
} catch (ArithmeticException | NullPointerException e) {  
    /* handle exception */  
5    e.printStackTrace();  
} catch (RuntimeException re) {  
    /* handle exception */  
    System.err.println("Runtime exception: " + e.getMessage());  
} catch (Throwable t) {  
10    /* handle any exception */  
    System.err.println("Unknown error has occurred");  
}
```

Analogicznie jak w przypadku konstrukcji **if-else**, jeśli dany wyjątek został już przechwycony w jednym z wcześniejszych bloków **catch** to nie będzie on przechwytywany przez pozostałe.

Dodatkowo, bezpośrednio po bloku **try** lub po blokach **catch**, można dodać blok **finally** pozwalający na zdefiniowanie instrukcji, które mają zostać zawsze wykonane po bloku **try**. Blok **finally** wykonywany jest zawsze – niezależnie od tego czy został zgłoszony wyjątek czy też nie. Tego typu funkcjonalność może być przydatna np. do zwalniania zasobów takich jak uchwyty do otwartych plików, połączenia sieciowe, itp. Przykładowo:

```
try {  
    /* do something that may fail */  
} finally {  
    /* cleanup */  
5 }
```

Do zgłaszania wyjątków służy słowo kluczowe **throw**, jak przedstawiono poniżej:

```
if (fail()) {  
    // something failed  
    throw new Exception("Something failed!");  
}
```

Po instrukcji **throw** podawany jest obiekt który zostanie zgłoszony – w tym przypadku jest to instancja klasy `Exception`. W tym miejscu (linia 3) natychmiast przerywany jest standardowy przebieg działania programu – do najbliższego odpowiedniego bloku **catch** lub **finally**. Jeśli odpowiedni blok **catch** nie został odnaleziony program kończy swoje działanie.

Zgłaszany może być wyjątek dowolnego typu implementującego interfejs `Throwable`, jednak zazwyczaj jest to instancja klasy dziedziczącej po `Exception`.

Dodatkowe informacje:

- Wyjątki – <https://docs.oracle.com/javase/tutorial/essential/exceptions/index.html>

Kolekcje

Na rysunku 1 przedstawiono zbiór interfejsów dla podstawowych kolekcji w Javie – będących rdzeniem Java Collections Framework. Interfejsy te pozwalają na korzystanie z kolekcji niezależnie od szczegółów ich implementacji. Jak można zauważyć na rysunku, interfejsy te tworzą hierarchię. `Set` jest specjalnym rodzajem `Collection`, `SortedSet` jest specjalnym rodzajem `Set`, itd.



Rysunek 1: Podstawowe interfejsy kolekcji.

- `Collection` – podstawa hierarchii kolekcji. Kolekcja reprezentuje grupę obiektów zwanych elementami. Jest najmniejszym wspólnym interfejsem, oraz pozwala na przekazywanie kolekcji gdy pożądanym jest najwyższy poziom abstrakcji.
- `Set` – zbiór posiadający wyłącznie unikalne elementy. Typowa implementacja: `HashSet`.
- `List` – uporządkowana kolekcja, która może posiadać wiele tych samych elementów. Typowa implementacja: `ArrayList`.
- `Queue` – kolejka zazwyczaj (choć nie zawsze) przechowująca elementy w kolejności FIFO (first-in, first-out). Niezależnie od stosowanej kolejności, głowa kolejki jest elementem, który zostanie usunięty podczas wywoływania operacji `remove` lub `poll`.
- `Deque` – kolejka, która może być używana zarówno jako FIFO (first-in, first-out) jak i LIFO (last-in, first-out) z tego względu, że możliwy jest dostęp do elementów z obu końców kolejki.
- `Map` – obiekt pozwalający mapować klucze na ich wartość. Nie może posiadać podwójnych kluczy, ale wiele kluczy może wskazywać na tę samą wartość. Typowa implementacja: `HashMap`.

Wszystkie podstawowe interfejsy kolekcji są generyczne (ang. generic). Przykładowo, tak wygląda deklaracja interfejsu `Collection`:

```
public interface Collection<E> ...
```

Składnia `<E>` pozwala na oznaczenie danego interfejsu (lub klasy) jako generyczną. Podczas deklaracji instancji `Collection` powinien zostać podany typ obiektów przechowywanych w kolekcji. Przykładowo, poniższy fragment pozwala zadeklarować dynamiczną listę liczb całkowitych:

```
Collection<Integer> intList = new ArrayList<>();
```

Taka specyfikacja pozwala na weryfikację (w trakcie kompilacji) typu elementów dodawanych do kolekcji – zmniejszając tym samym liczbę potencjalnych błędów w trakcie uruchomienia programu.

Dodatkowe informacje:

- Kolekcje – <https://docs.oracle.com/javase/tutorial/collections/>

Zadania

Zadanie 1

Zaprojektuj i zaimplementuj klasę `BinomialSolver` reprezentującą wielomian drugiego stopnia postaci $ax^2 + bx + c$ oraz pozwalającą na wyznaczenie pierwiastków równania kwadratowego postaci $ax^2 + bx + c = 0$. Każdy obiekt będący instancją klasy reprezentuje jeden wielomian, czyli powinien przechowywać wszystkie współczynniki (w polach prywatnych). Zaimplementuj poniższe funkcjonalności:

1. konstruktor przyjmujący wartości wszystkich współczynników (a , b oraz c),
2. szukanie pierwiastków równania kwadratowego $ax^2 + bx + c$ w dziedzinie liczb rzeczywistych, pierwiastki zapisz w osobnym polu/polach klasy,
3. metody dostępu – w trybie do odczytu – do parametrów wielomianu (np. `double getA()`) oraz wyliczonych pierwiastków równania (np. `double getX1()`),
4. wyznaczanie wartości wielomianu dla zadanej zmiennej x (np. metoda `double calculate(double x)`),
5. zgłaszanie wyjątku typu `ArithmeticException` (np. w konstruktorze) w przypadku, gdy równanie nie posiada rozwiązań z dziedziny liczb rzeczywistych.

Przetestuj działanie programu dla funkcji:

- $x^2 + 5x + 3$ – posiada dwa miejsca zerowe,
- $x^2 + 2x + 1$ – posiada jedno miejsce zerowe ($x_1 = x_2$),
- $6x^2 + 3x + 9$ – nie posiada miejsc zerowych.

Wskazówka 1 Obliczenia (wyznaczanie pierwiastków) możesz umieścić w konstruktorze lub wykonywać za pierwszym razem, gdy wywoływana jest jedna z metod `getX1()`/`getX2()`.

Zadanie 2*

Wykorzystując zadanie z poprzednich dotyczące figur, zaimplementuj sprawdzanie (w konstruktorze klasy `Rectangle`) czy podane wymiary są poprawnie. W przypadku podania nieprawidłowych wymiarów prostokąta (tj. mniejszych lub równych 0) zgłoś wyjątek `IllegalArgumentException` z odpowiednim komunikatem.

W programie głównym korzystając z konstrukcji **try-catch**, przetestuj działanie przygotowanych wyjątków dla klas `Rectangle` oraz `Square`. Obsłuż zgłaszane wyjątki i wyświetl użytkownikowi stosowny komunikat, a następnie podejmij odpowiednie działanie (np. ponownie zapytaj o wartość).

Zadanie 3

Wykorzystując kolekcję `ArrayList` zaimplementuj poniższą funkcjonalność:

- wylosuj n liczb całkowitych z przedziału $\langle 0; 10 \rangle$ i umieść je w kolekcji,
- z wykorzystaniem metody `get` wyświetl całą zawartość kolekcji na konsoli,
- z wykorzystaniem metody `contains` sprawdź czy kolekcja zawiera element wskazany przez użytkownika,
- z wykorzystaniem metody `remove` usuń element o indeksie wskazanym przez użytkownika,
- z wykorzystaniem iteratorów wyświetl całą zawartość kolekcji na konsoli.

Wskazówka 1 Możesz skorzystać z `For-Each Loop`.

Zadanie 4

Zaimplementuj funkcjonalność z zadania poprzedniego z wykorzystaniem `LinkedList`.

Zadanie 5

Wykorzystaj algorytmy standardowe `Collections.min` oraz `Collections.max` znajdź w kolekcjach z poprzednich zadań wartości największe i najmniejsze.

Zadanie 6

Wykorzystaj metodę `sort` do posortowania obu kolekcji zarówno w porządku rosnącym jak i malejącym.

Zadanie 7

Przekształć przygotowaną wcześniej listę w kolekcję posiadającą wyłącznie unikalne elementy (Set), a następnie wyświetl jej zawartość.

Wskazówka 1 Skorzystaj z odpowiedniego konstruktora klasy HashSet.

Zadanie 8*

Wykorzystując metodę `removeIf` usuń z listy elementy spełniające podany przez użytkownika warunek.

Zadanie 9*

Korzystając z implementacji klas `Osoba` oraz `Pracownik` z poprzednich zajęć, zdefiniuj klasę `Firma`, przechowującą spis wszystkich pracowników. W firmie nie powinno być dwóch pracowników o tym samym imieniu i nazwisku – do przechowywania obiektów skorzystaj z odpowiedniej implementacji interfejsu `java.util.Set`. Zaimplementuj metody:

- sprawdzając czy pracownik o podanym imieniu i nazwisku już istnieje,
- dodając nowego pracownika do firmy (zgłoś wyjątek typu `IllegalStateException` z odpowiednim komunikatem o błędzie gdyby dany pracownik był już wcześniej dodany),
- zwracającą liczbę wszystkich pracowników,
- usuającą pracownika o podanym imieniu i nazwisku, oraz
- wypisującą na ekran aktualny spis pracowników.

Rozdziel implementację odpowiednio pomiędzy poszczególne klasy.

Wskazówka Aby rozdzielić implementację pomiędzy poszczególne klasy, możesz przeciążyć metodę `toString()` w klasach `Pracownik` oraz `Osoba`, a następnie wykorzystać tę implementację w klasie `Firma`.

Zadanie 10*

W klasie `Firma` zaimplementuj metodę `iterator()` tak aby można było przeglądać kolekcję wszystkich pracowników firmy z wykorzystaniem pętli `for-each` w następujący sposób:

```
Firma firma = new Firma();  
// ...  
for (Pracownik p : firma) {  
    // ..  
}
```

5

Wskazówka 1 Klasa `Firma` powinna implementować interfejs `java.lang.Iterable`.

Wskazówka 2 Możesz skorzystać z implementacji iteratora kolekcji pracowników w klasie `Firma (Collection#iterator())`.