Programowanie obiektowe

Zajęcia 6. Wyjątki

Zadanie 1. Interfejsy i wyjątki Java

1. Utwórz w wybranym miejscu na dysku folder `WyjatkiJava`.
2. Utwórz klasę `Customer`, która będzie zawierała podstawowe informacje o kliencie banku takie jak np. nazwisko, a także inne, zaproponowane samodzielnie (np. adres, numer PESEL).

```
public class Customer {
        private String name;

        public Customer(){}
        public Customer(String s){
    //ciało metody
        }
        public String getName() {
    //ciało metody
        }

        @Override
        public boolean equals(Object c) {
    //ciało metody
        }
}
```

3. Napisz ciała konstruktorów dla powyższej klasy, a także odpowiednich metod. Przesłoń metodę `equals` i zdefiniuj w niej porównanie klientów

z wykorzystaniem ich nazwiska (metoda ma zwracać `true` gdy nazwiska są identyczne, `false` w przeciwnym przypadku). Napisz też krótki komentarz związany z tym, do czego służy adnotacja `@Override`.

4. Utwórz klasę `Account`, która będzie zawierała podstawowe informacje dotyczące konta w banku, takie jak dane logowania, numer konta, właściciel konta, stan konta. Zwróć uwagę, że jedna osoba może być właścicielem kilku kont, ale dane konto należy tylko do jednej osoby.

```
public class Account {
  private int accountNumber;
  private Customer owner;
  private double balance;
  private int password;
  private String login;

  public Account(){}
  public Account(int n, Customer c, int p, String l){
    //ciało metody
  }
  public void login(String l, int passwd)
        throws AccountLoginFailedException {
    //ciało metody
  }
  public Customer getCustomer(){
    //ciało metody
  }
  public int getNumber() {
    //ciało metody
  }
  public double getBalance() {
    //ciało metody
  }
  public double withdraw(double amount) {
    //ciało metody
  }
}
```

5. Wykonaj implementację odpowiednich metod dla klasy.

6. Utwórz własną klasę wyjątku `AccountLoginFailedException` związaną z obsługą błędów logowania.

```
class AccountLoginFailedException extends Exception{
  static final long serialVersionUID = 1L;
  private int password;
  private String login;
  //pola odpowiadające danym klienta oraz numerowi konta
```

```
  //przykładowe konstruktory
  AccountLoginFailedException(){}
  AccountLoginFailedException(String errorMsg, String s, int passwd){
    super(errorMsg);
    login = s;
    password = passwd;
  }
  //przykładowa metoda
  int getPassword(){
    //ciało metody
  }
}
```

Stwórz konstruktor bezparametrowy oraz odpowiedni konstruktor(y) parametrowy umożliwiający podanie danych osoby, która próbowała się zalogować oraz jej dane logowania. Napisz odpowiednie metody pozwalające odczytać te dane.

7. Zdefiniuj następujące interfejsy.

```
interface SearchCustomers {
    Customer findByName(String name)
                throws CustomerNotFoundException;
}

interface SearchAccounts {
    Account findByNumber(int number)
                throws AccountNotFoundException;
    ArrayList<Account> findAllCustomerAccounts(Customer cust)
                throws AccountNotFoundException;
}
```

Metoda `findByName` interfejsu `SearchCustomers` pozwala na znalezienie klienta banku po nazwisku. W razie gdy bank nie ma takiego klienta, zgłasza wyjątek `CustomerNotFoundException`. Metoda `findByNumber` interfejsu `SearchAccounts` pozwala na znalezienie konta w banku po jego numerze. W razie gdy takie konto w banku nie istnieje, zgłasza wyjątek `AccountNotFoundException`. Metoda `findAllCustomerAccounts` interfejsu `SearchAccounts` pozwala na znalezienie wszystkich kont danej osoby, które ma ona w banku. W razie gdy klient nie posiada jeszcze żadnego konta w banku zgłaszany jest wyjątek `AccountNotFoundException`. Należy utworzyć własne klasy wyjątków `CustomerNotFoundException` oraz `AccountNotFoundException` wraz z odpowiednimi metodami.

8. Utwórz klasę `Bank`, która implementuje powyższe interfejsy (podaj implementację ich metod):

```
public class Bank implements SearchAccounts, SearchCustomers {
  ArrayList<Customer> customers = new ArrayList<Customer>();
  ArrayList<Account> accounts = new ArrayList<Account>();

  @Override
  public Customer findByName(String name)
                throws CustomerNotFoundException {
    //ciało metody
  }
  @Override
  public Account findByNumber(int num)
                throws AccountNotFoundException {
    //ciało metody
  }
  @Override
  public ArrayList<Account> findAllCustomerAccounts(Customer cust)
                throws AccountNotFoundException {
    //ciało metody
  }
}
```

W ramach klasy dodaj też implementację kilku metod, np. dodanie klienta banku, dodanie nowego konta etc.

9. Zastanów się co zyskujemy stosując interfejsy, napisz krótki komentarz na ten temat.

10. W funkcji `main` przetestuj podstawową funkcjonalność zaimplementowanych klas, np. dodawanie nowego konta, dodawanie nowego klienta banku, sprawdzenie czy dane konto istnieje, sprawdzenie czy dana osoba jest klientem banku, logowanie, wyszukanie wszystkich kont danego klienta banku. Otocz wywołania metod blokiem `try-catch` zawierającym po jednym bloku `catch` dla każdego z wyjątków.

11. Po przechwyceniu wyjątku w bloku `catch`, wypisz na standardowe wyjście ślad stosu wywołań z chwili zgłoszenia wyjątku (metoda `printStackTrace()`). Bezpośrednio po wywołaniu metody `printStackTrace()` dla wyjątku, zgłoś obsługiwany wyjątek ponownie. Czy ślady stosu wypisane w bloku `catch` i przez maszynę wirtualną w chwili przerwania programu są takie same? (Zadanie wykonaj dla jednego, wybranego wyjątku np. `CustomerDataNotFoundException`).

12. Zmodyfikuj kod tak, aby przed ponownym zgłoszeniem wyjątku w bloku `catch` została wykonana dla niego metoda `fillInStackTrace()`. Czy ślady stosu wypisane w bloku `catch` i przez maszynę wirtualną w chwili przerwania programu są takie same? Jeśli nie, to dlaczego.

Napisz krótki komentarz na ten temat. (Zadanie wykonaj dla jednego, wybranego wyjątku np. `CustomerDataNotFoundException`).

Zadanie 2. Wyjątki w C++

1. Utwórz w wybranym miejscu na dysku folder `WyjatkiCpp`.

2. Utwórz klasę `Stack`, która będzie implementacją stosu do przechowywania elementów typu `int`. Stwórz konstruktor bezparametrowy ustawiający domyślne wartości paramatrów oraz konstruktor przyjmujący jako argument rozmiar stosu. Zaimplementuj metody dodawania i pobierania elementów ze stosu. Metoda `push` powinna zgłaszać wyjątek gdy został przekroczony maksymalny rozmiar stosu. Metoda `pop` powinna zgłaszać wyjątek, gdy stos jest pusty.

```
class Stack{
private:
  int maxStackSize;
  vector<int> dfs;
  int top;
public:
  Stack();
  Stack(int max);
  void push(int newItem);
  int pop();
};
```

3. Utwórz następujące klasy wyjątków:

```
class StackException : public exception{
public:
  StackException() {}
  virtual const char* what() const noexcept override{
    return "Bledna operacja na stosie!";
  }
};

class StackFullException : public StackException{
  int element = 0;
  int maxStackSize = 0;
  string message;
public:
  StackFullException(){}
  StackFullException(string p, int e, int s){
    //cialo metody
  }
  virtual const char* what() const noexcept override {
    //cialo metody - komunikat o błędzie
```

```
    //wraz informacją o rozmiarze stosu i nadmiarowym elemencie
  }
};

class StackEmptyException : public StackException{
  string message;
public:
  StackEmptyException(){}
  StackEmptyException(string p){
    //ciało metody
  }
  virtual const char* what() const noexcept override{
    //ciało metody
  }
};
```

Powyższy kod używa operatora `noexcept`, napisz krótki komentarz dlaczego. Co daje nam zastosowanie modyfikatora `override` i jak ma się to do adnotacji `@Override` w Javie? Napisz krótki komentarz na ten temat.

4. Zauważ, że w deklaracjach metod klasy `Stack` nie ma specyfikacji wyjątków, które mogą one zgłosić. Napisz krótki komentarz dlaczego.

5. Utwórz funkcję przyjmującą obiekt klasy `Stack` przez referencję i wywołaj ją. Np.:

```
void foo(Stack &s) {
  try{
    //implementacja
  } catch (StackException e){
    cout << e.what() << endl;
  }
}
```

W ramach tej funkcji dodawaj kolejne elementy do stosu tak długo, aż zostanie przekroczony jego zadeklarowany rozmiar. Zapisz swoje obserwacje i spróbuj je zinterpretować.

6. Utwórz funkcję przyjmującą przez referencję obiekt klasy `Stack` i wywołaj ją. Np.:

```
void bar(Stack &s) {
  try{
    //implementacja
  } catch (StackException &e){
    cout << e.what() << endl;
  }
}
```

W ramach tej funkcji dodawaj kolejne elementy do stosu tak długo, aż zostanie przekroczony jego zadeklarowany rozmiar. Zapisz swoje obserwacje i spróbuj je zinterpretować odnosząc je do wyników z poprzedniego punktu.

7. W ramach funkcji `main` zaimplementuj podstawową interakcję z użytkownikiem w zakresie dodawania/usuwania elementów ze stosu. Minimalizuj użycie bloków `try`, w jednym takim bloku można przechwytywać i obsługiwać wiele różnych wyjątków.

```
//...
  try{
    //implementacja interakcji z użytkownikiem
  } catch(StackException &e){
    //...
  } catch(exception &e){
    //...
  } catch(...){
    //...
  }
//...
```