Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря
Сікорського”Факультет інформатики та
обчислювальної техніки Кафедра інформаційних
систем та технологій

Лабораторна робота №13

із дисципліни «Основи програмування»

Тема: «Exceptions»

Виконали:
Студенти групи ІА-24
Красношапка Роман Олександрович
Орловська Анна Валеріївна
Бакалець Андрій Ігорович
Перевірив:
Колесник Валерій Миколайович

Київ-2023

Хід роботи:
1. Ознайомитись з javadoc для наступних класів:
   - Throwable;
   - Error;
   - Exception;
   - RuntimeException;

а також повторити конструкції try-catch-finally, throw, throws.

2. Проаналізувати предметну область з л/р 10-12. Створити свій власний тип checked exception (підклас класу Exception), що описує порушення одного з бізнес-правил предметної області (наприклад, спробу додати студента у групу, в якій вже навчається максимально допустима кількість студентів, або пуста "" назва фільму). Додати throws-декларацію цього exception до відповідних методів, у яких потрібно «викидати» його у разі порушення бізнес-правил. Додати блок try-catch-finally для демонстрації виклику цього метода. Продемонструвати «викидання» та обробку цього exception. Також додати та продемонструвати код для «викидання» кількох стандартних RuntimeException (IllegalArgumentException, NullPointerException, ...).

3. Відповісти на контрольні питання.

Код програми:

```
import java.io.IOException;
import java.util.*;

public class Main {
    public static void main(String[] args) throws RuntimeException {
        printResult();
    }


    public static void printResult() throws IllegalArgumentException {

        Film film1 = new Film("Deadpool");
        Film film2 = new Film("Focus");
        Film film3 = new Film("The Pursuit of Happyness");
        Actor actor1 = new Actor("Ryan Reynolds");
        Actor actor2 = new Actor("Will Smith");
        Actor actor3 = new Actor("Syre Smith");
        Actor actor4 = new Actor("Brad Pitt");

        try {
            film2.setName("");
        } catch (InvalidFilmNameException e) {
            System.out.println("Error: " + e.getMessage());
            return;
        }
        try {
            actor1.setActor("");
        } catch (InvalidActorNameException e) {
            throw new RuntimeException(e);
        }

        Database.addFilm(film1);
        Database.addFilm(film2);
        Database.addFilm(film3);
        Database.addActor(actor1);
        Database.addActor(actor2);
        Database.addActor(actor3);
```

```
        Database.addActor(actor4);

        film1.addActor(actor1);
        film2.addActor(actor2);
        film3.addActor(actor2);
        film3.addActor(actor3);
        actor1.addFilm(film1);
        actor2.addFilm(film2);
        actor2.addFilm(film3);
        actor3.addFilm(film3);

        System.out.println(getCoActors(actor2));
        System.out.println("The largest number of actors in the film - " +
mostActors());
        notFilmed();

    }

    private static String mostActors() throws RuntimeException {
        int countActor = 0;
        String mostActored = "";
        List<Film> allfilms = Database.getFilms();
        for (Film film : allfilms) {
            if (countActor < film.getActors().size()) {
                countActor = film.getActors().size();
                mostActored = film.getName();
            } else if (countActor == film.getActors().size() &&
film.getActors().size() != 0) {
                mostActored = film.getName() + " and " + mostActored;
            }
        }
        return mostActored;
    }


    private static void notFilmed() throws NullPointerException {
        List<Actor> allActors = Database.getActors();

        for (Iterator<Actor> i = allActors.iterator(); i.hasNext(); ) {
            Actor actor = (Actor) i.next();
            if (actor.getFilms().isEmpty()) {
                System.out.println(actor.getName() + " - didn't act in any film ");
            }
        }
    }

    public static String getCoActors(Actor actor) throws NullPointerException {
        List<Actor> coActors = new ArrayList<>();
        List<Film> films = Database.getFilms();

        Iterator<Film> filmIter = films.iterator();
        while (filmIter.hasNext()) {
            Film film = filmIter.next();
            if (film.getActors().contains(actor)) {
                List<Actor> actors = film.getActors();
                Iterator<Actor> actorIter = actors.iterator();
                while (actorIter.hasNext()) {
                    Actor coActor = actorIter.next();
                    if (!coActor.equals(actor) && !coActors.contains(coActor)) {
                        coActors.add(coActor);
                    }
                }
            }
        }
        StringBuilder sb = new StringBuilder();
        sb.append("Co-actors of ").append(actor.getName()).append(": ");

        Iterator<Actor> coActorIter = coActors.iterator();
```

```
        while (coActorIter.hasNext()) {
            Actor coActor = coActorIter.next();
            sb.append(coActor.getName()).append(", ");
        }
        if (coActors.size() > 0) {
            sb.delete(sb.length() - 2, sb.length());
        }
        return sb.toString();
    }
}
```

```
import java.util.ArrayList;
import java.util.List;

public class Actor{
    private String name;
    private final List<Film> films = new ArrayList<>();
    public Actor(String name) {
        this.name = name;
    }

    public void setActor(String name) throws InvalidActorNameException {
        if (name == null || name.trim().isEmpty()) {
            throw new InvalidActorNameException("Invalid film name: " + name);
        }
        this.name = name;
    }

    public void addFilm(Film film) {
        this.films.add(film);
    }
    public List<Film> getFilms(){
        return this.films;
    }
    public String getName() {
        return this.name;
    }

}
class InvalidActorNameException extends Exception {

    public InvalidActorNameException(String message) {
        super(message);
    }

}
```

```
import java.util.ArrayList;
import java.util.List;

class Database {
    private static final List<Film> allFilms = new ArrayList<>();
    public static void addFilm(Film film) {
        allFilms.add(film);
    }

    private static final List<Actor> allActors = new ArrayList<>();
    public static void addActor(Actor actor) {
        allActors.add(actor);
    }

    public static List<Actor> getActors(){
        return allActors;
    }
    public static List<Film> getFilms(){
```

```
        return allFilms;
    }
}
```

```
import java.util.ArrayList;
import java.util.List;

public class Film{
    private String name;
    private final List<Actor> actors = new ArrayList<>();
    public Film(String name) {
        this.name = name;
    }
    public void setName(String name) throws InvalidFilmNameException {
        if (name == null || name.trim().isEmpty()) {
            throw new InvalidFilmNameException("Invalid film name: " + name);
        }
        this.name = name;
    }

    public void addActor(Actor actor) {
        this.actors.add(actor);
    }
    public List<Actor> getActors(){
        return this.actors;
    }
    public String getName() {
        return this.name;
    }
}
class InvalidFilmNameException extends Exception {

    public InvalidFilmNameException(String message) {
        super(message);
    }

}
```

Результат:

```
Error: Invalid film name:
```

```
Exception in thread "main" java.lang.RuntimeException Create breakpoint : InvalidActorNameException: Invalid film name:
    at Main.printResult(Main.java:29)
    at Main.main(Main.java:6)
Caused by: InvalidActorNameException: Invalid film name:
    at Actor.setActor(Actor.java:13)
    at Main.printResult(Main.java:27)
    ... 1 more
```

```
Co-actors of Will Smith: Syre Smith
The largest number of actors in the film - The Pursuit of Happyness
Brad Pitt - didn't act in any film
```

Висновок: Під час виконання даної лабораторної роботи було створено свій власний тип checked exception, що описує порушення бізнес-правил предметної області. Для цього було створено підклас класу Exception та додано throws-декларацію до відповідних методів, у яких може виникнути цей виняток. Також був продемонстрований блок try-catch-finally для обробки цього винятку та додатково були продемонстровані кілька стандартних RuntimeException, таких як IllegalArgumentException та NullPointerException. Використання винятків є важливим інструментом у програмуванні, що дозволяє виявити та обробити помилки в програмі. Власний тип винятку може бути створений для опису конкретних помилок, які виникають у предметній області. Розуміння використання винятків та їх обробка є важливим навичком для кожного програміста.