­­МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

­­­

Лабораторна робота №5

з курсу «Об’єктно-орієнтоване програмування»

для студентів базового напрямку 6.08.04 "Комп’ютерні науки"

(заочна форма навчання)

Варіант 14

Виконав студент гр. КНз-2

Чалий Михайло

­­

Львів 2014

# Обробка виняткових ситуацій

## Мета роботи

Одержати навики роботи із оброблення виняткових ситуацій.

## Теоретичні відомості

Виняткові ситуації (exceptions) можуть виникнути під час виконання (runtime) програми, перервавши її звичайний хід. До них відносится ділення на нуль, відсутність завантажуваного файла, індекс масива який став від’ємним або вийшов за верхню межу, переповнення виділеної пам’яті і багато інших неприємностей. Звичайно, можна передбачити такі ситуації і застрахуватися від них, наприклад таким чином:

if (something == wrong){

// Робимо аварійні дії

}else{

// Звичайний хід дій

}

Але при цьому багато часу іде на перевірки, і програма перетворюється у набір таких перевірок. Якщо переглянути промислову програму, написану мовою С або Pascal, то можна побачити, що вона на 2/3 складається із таких перевірок. В об’єктно-орієнтованих мовах програмування прийнято інший підхід. При виникненні виняткової ситуації виконуюча система створює об’єкт певного класу, який відповідає ситуації, що виникла і який містить дані про те, що, де і коли трапилося. Цей об’єкт передається на обробку програмі, в якій виникла виняткова ситуація. Якщо програма не обробляє виняткову ситуацію, тоді об’єкт за замовчуванням повідомляє обробнику виконуючої системи. Обробник поступає дуже просто: виводить на консоль повідомлення про виянткову ситуацію і зупиняє виконання програми.

Приклад. У програмі лістингe 1 може виникнути ділення на нуль, якщо запустити її з аргументом 0. E програмі немає жодних засобів обробки такої виняткової ситуації.

Лістинг 1. Програма без обробки виняткових ситуацій

class SimpleExt {

public static void main(String[] args){

int n = Integer.parseInt(args[0]);

System.out.println("10 / n = " + (10 / n));

System.out.println("After all actions");

}

}

Програма SimpleExt запущена три рази. Перший раз аргумент args[0] дорівнює 5 і програма виводить результат: "10 / n = 2". Після цього з’являється друге повідомлення: "After all actions". Другий раз аргумент дорівнює 0, і замість результата ми отримуємо повідомлення про те, що в підпроцесі "main" відбулося виняткова ситуація у класі ArithmeticException внаслідок ділення на нуль: "/ by zero". Далі уточнюється, що виняткова ситуація виникло під час виконання метода main класу SimpleExt, а в дужках вказано, що дія, в результаті якої виникла виняткова ситуація, записана у четвертому рядку файла SimpleExt.java. Виконання програми зупиняється, заключне повідомлення не з’являється. Третій раз программа запущена взагалі без аргумента. В масиві args немає елементів, його довжина дорівнює нулю, і ми намагаємося звернутися до елемента args[0]. Виникає виняткова ситуація в класі ArrayIndexOutOfBoundsException внаслідок дії, записаної у третьому рядку файла SimpleExt.java. Виконання програми зупиняється, звернення до методу println() не відбувається.

### Блоки перехоплення виняткової ситуації

Ми можемо перехопити і обробити виняткову ситуацію у програмі. При описуванні обробки застосовується бейсбольна термінологія. Говорять, що виконуюча система або програма "викидає" (throws) об’єкт-виключення. Цей об’єкт "пролітає" через всю програму, з’явившись спочатку у тому методі, де відбулася виняткова ситуація, а програма в одному або декількох місцях намагається (try) її "перехопити" (catch) і обробити. Оброблення можна зробити повністю в одному місці, а можна обробити виняткову ситуацію в одному місці, викинути знову, перехопити у другому місці і обробляти далі.

Добре написані об’єктно-орієнтовані програми обовязково повинні обробляти всі виникаючі в них виняткові ситуації. Для того щоб спробувати (try) перехопити (catch) об’єкт-виняток, потрібно весь код програми, у якому може виникнути виняткова ситуація, охопити оператором try {} catch() {}. Кожний блок catch(){} перехоплює виняткову ситуацію тільки одного типу, вказаного в його аргументі. Але можна написати декілька блоків catch(){} для перехоплення декількох типів виняткових ситуацій. Наприклад, у програмі лістинга 2.1 можуть виникнути виняткові ситуації двох типів. Напишемо блоки їх обробки, як це зроблено в лістингу 2.

Лістинг 2. Програма з блоками обробки виключень

class SimpleExt1{

public static void main(String[] args){

try{

int n = Integer.parseInt(args[0]);

System.out.println("After parseInt()");

System.out.println(" 10 / n = " + (10 / n) ) ;

System. out. println ("After results output");

}catch(ArithmeticException ae){

System.out.println("From Arithm.Exc. catch: "+ae);

}catch(ArrayIndexOutOfBoundsException arre){

System.out.println("From Array.Exc.catch: "+arre);

}finally{

System.out.println("From finally");

}

System.out.println("After all actions");

}

}

У програму лістинга 2 вставлено блок try{} і два блоки перехоплення catch(){} для кожного типу виняткових ситуацій. Тут обробка виняткових ситуацій полягає просто у виведенні повідомлення і вмісту об’єкта-виключення, як воно представлено методом toString() відповідного класу-виключення. Після блоків перехоплення вставлено ще один, необов’язковий блок finally(). Він призначений для виконання дій, які треба виконати обов’язково, щоб не трапилося. Все, що написано у цьому блоці, буде виконано і при виникненні виняткової ситуації, і при звичайному ході програми, і навіть якщо вихід із блоку try{} здійснюється оператором return.

Якщо в операторі обробки виняткових ситуацій є блок finally{}, тоді блок catch() {} може бути відсутнім, тобто можна не перехоплювати виняткову ситуацію, але при її появі виконає певні обов’язкові дії. Крім блоків перехоплення у лістингу 2.2 після кожної дії робиться трасуючий друк, щоб можна було прослідкувати за порядком виконання програми.

## Висновки

Одержав навики роботи із оброблення виняткових ситуацій.