Лабораторна робота № 5

Тема: Використання та реалізація виключень

Мета: Вивчення призначення та принципів функціювання виключень

Теоретична частина

Виключення є винятковою ситуацією, що відбувається під час виконання певної послідовності програмного коду. Як правило, виключенням є помилка, що виникає під час виконання програми.

Якщо б не підтримувалась обробка виключень, то слід було б передбачати в коді програми перевірку і обробку помилок. Обробка виключень в Java дозволяє уникнути певних складностей з їх організацією, а крім того, переносить управління помилками, що виникають під час виконання, в область ООП.

Виключення в Java — це об’єкт, який описує виключну (помилкову) ситуацію, що виникає у певній частині програмного коду. В методі, що викликав помилку, генерується об’єкт, який і є виключенням. Цей метод може обробити виключення самостійно або ігнорувати його. У певний момент виключення перехоплюється та обробляється. Виключення можуть генеруватись автоматично виконавчою системою Java або вручну у прикладному коді. Виключення, що генеруються виконавчою системою Java, мають відношення до фундаментальних помилок, що порушують правила мови Java або обмеження, які накладаються виконавчою системою Java. Виключення, що генеруються вручну, зазвичай слугують для повідомлення коду, що його створює, про деякі помилки в методі.

Управління обробкою виключень в Java здійснюється за допомогою п’яти ключових слів:

try,
catch
throw
throws
finally

Оператори програми, які необхідно відстежувати на предмет виключень, розміщуються у блоці try{ ..... }.

Якщо виключення виникає в блоці try, то воно генерується.

Прикладний код може перехоплювати виключення (об’єкт-виключення), для чого використовується блок catch{ ..... }, де організовується його обробка.

Системні виключення автоматично генеруються виконавчою системою Java.

Для генерування виключення вручну використовується ключове слово throw. Будь-яке виключення, що генерується в тілі метода, повинно бути позначено в його оголошені ключовим словом throws.

Любий код, який при любому випадку необхідно виконати по завершені блоку try, розміщують у блоці finally.

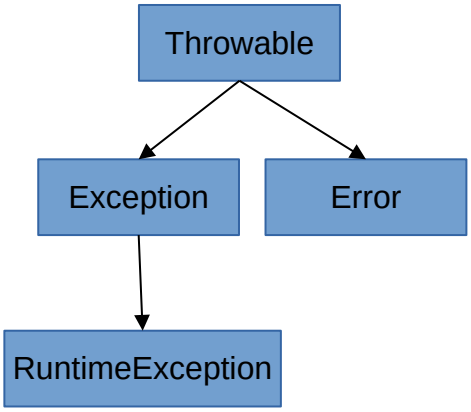
Загальна форма блоку обробки виключень:

```
try
{
 // блок коду, у якому відстежуються помилки
}
catch (тип_виключення_1 ex0b) {
// обробка виключень типу тип_виключення_1
}
catch (тип_виключення_2 ex0b) {
// обробка виключень типу тип_виключення_2
}
// ...
finally {
 // блок коду, який обов’язково буде виконано
}
```

Усі типи виключень є підкласами, похідними від вбудованого класу Throwable. Два його підкласи Exception та Error. Exception слугує для виключних умов, які повинна перехопити прикладна програма. При створені власних типів виключень свої підкласи наслідують клас Exception.

Підклас RuntimeException призначено для виключень, що автоматично визначаються для створюваних прикладних програм і охоплюють такі помилки, як ділення на нуль та помилкова індексація масивів.

Гілка класу Error, визначає виключення, поява яких не повинна бути при нормальному виконанні програми. Виключення типу Error використовуються у виконавчій системі Java для позначення помилок, які відбуваються у самому виконавчому середовищі. Прикладом такого роду помилок є переповнення стеку. Як правило, такого роду виключення не призначено для обробки прикладною програмою, тому що вони виникають при аварійних ситуаціях.



Приклад програми, в якій є оператор, що викликає помилку ділення на нуль.

```
public class Exc0 {
   public static void main(String args[]){
      int d = 0;
      int a = 42 / d;
   }
}
```

Коли виконавча система Java виявляє спробу ділення на нуль, вона створює новий об’єкт виключення, а потім генерує саме виключення, яке і припиняє виконання класу Exc0. Це відбувається тому, що як тільки виключення згенеровано, то воно повинно бути перехопленим обробником виключень та оброблено їм. У даному випадку обробник у програмі відсутній і тому виключення перехоплюються стандартним обробником виконавчої системи Java. Будь-яке виключення, що не перехоплюється прикладною програмою, перехоплюється та оброблюється стандартним обробником.

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at newpackage.Exc0.main(Exc0.java:19)
```

Згенероване виключення відноситься до підкласу ArithmeticException, що є похідним від класу Exception і точно описує тип помилки, що виникла.

Трасування стеку дозволяє визначити послідовність викликів методів, які привели до помилки. Модифікуємо попередню програму таким чином:

```
public class Exc0 {
   static void subr(){
      int d = 0;
      int a = 42 / d;
   }
   public static void main(String[] args) {
      Exc0.subr();
   }
}
```

Результат трасування стека стандартного обробника виключень буде тепер таким:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at newpackage.Exc0.subr(Exc0.java:19)
        at newpackage.Exc0.main(Exc0.java:23)
```

Метод, що викликав помилку

Метод, що викликав метод з помилкою

Використання блоку try{}catch{} надає можливість виправити помилку, а також унеможливлює автоматичне переривання виконання програми.

Приклад 1

```
class Example {
  public static void main(String args[]) {
    int d, a;

    try { // контролювання блоку коду
      d = 0;
      a = 42 / d;
      System.out.println("This will not be printed.");
    } catch (ArithmeticException e) { // перехоплення об’єкту виключення
      System.out.println("Division by zero.");
    }
    System.out.println("After catch statement.");
  }
}
```

Оператори try можуть бути вкладеними. Один оператор try може знаходитись у блоці іншого оператору try. Кожного разу, коли управління передається блоку оператора try, контекст відповідного виключення розміщується у стеку. Якщо у вкладеному операторі try відсутній оператор catch для перехоплення і обробки конкретного виключення, стек розгортається і перевіряється на відповідний оператор catch із зовнішнього блоку оператора try, і так робиться до тих пір, поки не буде знайдено відповідний оператор catch або не будуть вичерпані усі рівні вкладеності операторів try. Якщо відповідний оператор catch не буде знайдено, то створене виключення оброблюється виконуючою системою Java.

Приклад 2

```
class Example {
  public static void main(String args[]) {
    try {
      int a = args.length;
      int b = 42 / a;  /* Генерується виключення - a divide-by-zero */
      System.out.println("a = " + a);
      try {
           if(a==1) a = a/(a-a); // поділ на нуль
           if(a==2) {
                 int c[] = { 1 };
                 c[42] = 99; // генерація виключення - out-of-bounds
            } // блок if
          } catch(ArrayIndexOutOfBoundsException e) {
```

```
                System.out.println("Індекс масиву за межами: " + e);
            }
        } catch(ArithmeticException e) {
                System.out.println("Ділення на 0: " + e);
        }
    }
}
```

Виключення можна генерувати безпосередньо в прикладній програмі за допомогою оператору throw. Його загальна форма виглядає наступним чином:

```
throw екземпляр_генерації;
```

Екземпляр, що генерується, повинен бути об’єктом класу Throwable або похідного від нього підкласу. Примітивні типи на кшталт int або char, а також класи, крім Throwable, наприклад, String або Object, неможна використовувати для генерування виключень. Отримати об’єкт класу Throwable можна або вказуванням відповідного параметру в операторі catch або створенням цього об’єкту за допомогою оператору new.

Потік виконання програми зупиняється після оператору throw, і всі наступні оператори не виконуються. Найближчій охоплювальний блок оператора try перевіряється на наявність оператора catch із співпадаючим типом виключення. Якщо співпадіння виявленно, то управління передається цьому оператору. У іншому випадку перевіряється наступний зовнішній блок оператора try і т.д. Якщо не вдається знайти оператор catch, що співпадає з типом виключення, то стандартний обробник виключень прериває виконання програми і виводе результат трасування стеку.

Приклад 3

```
class ThrowDemo {
  static void demoproc() {
    try {
      throw new NullPointerException("demo");
    }
    catch(NullPointerException e) {
      System.out.println("Виключення перехоплено в методі demoproc()");
      throw e; // повторне генерування виключення
    }
  }

  public static void main(String args[]) {
    try {
      demoproc();
    } catch(NullPointerException e) {
```

```
            System.out.println("В main() перехоплено виключення: " + e);
        }
      }
    }
```

Якщо метод здатний викликати виключення, яке він сам не обробляє, тоді він повинен задати свою поведінку таким чином, щоб код, який буде викликати цей метод міг захистити себе від такого виключення. Задля цього до оголошення методу додається оператор throws, де перераховуються типи виключень, які метод може генерувати. Це є обов'язковим для усіх виключень, крім тих, що відносяться до класів Error та RuntimeException або будь-яким їх підкласам. Загальна форма методу, що має оператор throws:

```
тип ім'я_методу(список параметрів) throws список_виключень
        {
            // тіло методу
        }
```

Приклад 4

```
public class Example {
   static void throwOne() throws IllegalAccessException {
      System.out.println("У середені методу throwOne() ");
      throw new IllegalAccessException(" демонстрація " );
   }
   public static void main (String args[]){
      try{
         throwOne();
      }catch(IllegalAccessException e){
         System.out.println("Exception:"+e);
      }
   }
}
```

Завдання до лабораторної роботи

1. Вивчить теоретичний матеріал та повторіть наведені приклади.
2. Наведіть скриншоти отриманих результатів та надайте пояснення.
3. Підготуйте та надайте звіт.

Контрольні запитання

1. Що таке виключна ситуація?
2. Якими способами можуть генеруватись виключення?
3. В яких випадках викликається стандартний обробник виключень Java?
4. Які ключові слова використовуються для обробки виключень в Java?
5. Яке призначення конструкції try… catch…finally?