1 ОПРАЦЮВАННЯ ВИНЯТКОВИХ СИТУАЦІЙ

1.1 СУТНІСТЬ ОПРАЦЮВАННЯ ВИНЯТКОВИХ СИТУАЦІЙ

Виняткова ситуація в програмуванні — це подія, яка виникає у процесі виконання програми і може змінити подальший хід її виконання. Причини таких подій можуть бути різні, але найчастіше вони виникають у зв'язку з наявністю помилок у програмі, наприклад, ділення на нуль, звернення до елемента масиву з індексом, якого у масиві не існує. Коли виникають такі винятки, програма припиняє своє виконання, і на екран видається відповідне повідомлення. Але у мові Java є засоби, які дозволяють перехопити виняток, опрацювати його, продовжити виконання програми до її завершення і висвітлити на екрані повідомлення про наявність виняткової ситуації.

Якщо у програмі мовою Java виникає виняток, то в методі, у якому вона виникла, автоматично створюється об'єкт, що описує цей виняток. Цей об'єкт передається для опрацювання методу (викидається в метод), у якому виник виняток. Виняток може бути опрацьований одразу в методі або переданий далі для опрацьовування. Але в решті-решт його буде перехоплено й опрацьовано.

Для того щоб метод зміг опрацьовувати виняткову ситуацію, необхідно у програмному коді передбачити її опрацьовування. Для цього необхідно виділити фрагмент коду, який слід контролювати на предмет появи винятку, а також передбачити необхідний клас його опрацювання.

Виняток може генеруватися автоматично виконавчою системою Java, або його генерування може бути передбачено програмістом у програмному коді спеціальними програмними методами. Ті винятки, що генеруються виконавчою системою Java, відносять до категорії фундаментальних. Вони можуть виникати, наприклад, у разі порушення синтаксичної конструкції мови або встановлених обмежень. Винятки, що передбачаються програмістом у коді, слугують, зазвичай, для повідомлення про помилки в методі, що викликається, і продовження виконання коду.

Для управління опрацюванням винятків використовується блок **try-catch-finally**. Сутність опрацювання винятків така: сукупність команд програми, у якій потрібно відслідковувати винятки, розміщується у блоці try. У прикладному коді передбачається також один або декілька блоків catch. Кожний блок catch спроможний перехопити й опрацювати не будь-яку помилку, а лише ту, тип якої вказано у блоці try. Таким типом може бути, наприклад, вихід індексів масиву за межі допустимих значень. Якщо в блоці try виникає помилка,

управління перехоплюється тим блоком catch, який спроможний опрацювати саме цю помилку.

Отже, у блоці catch опрацьовується не будь-яка виняткова ситуація, а лише та, яка є параметром слова catch. Відповідний параметр указується після слова catch у круглих дужках. Якщо блоків catch декілька, то під час виникнення винятку вони переглядаються послідовно до тих пір, поки тип винятку не співпаде з аргументом у блоці catch. Після завершення виконання операторів блоку catch управління буде передано команді, що слідує після всього блоку операторів try-catch.

Нагадаємо, що системні винятки генеруються автоматично виконавчою системою Java. Для генерування винятків вручну використовується ключове слово throw. А будь-який випадок, що генерується у тілі методу, необхідно визначити в його оголошенні за допомогою ключового слова throws. Після блоків try і catch можна вказати блок finally. У ньому розміщуються оператори, які повинні виконуватися обов'язково після завершення блоку try. Отже, загальна структура блоку опрацювання винятків така:

```
try {
//код, у якому відслідковується і генерується виняток
}
catch (тип винятку_1 об'єкт) {
//код опрацювання винятку_1
}
catch (тип винятку_2 об'єкт) {
//код опрацювання винятку_2
}
//…
finally {
//код, що виконується обов'язково після завершення блоку try-catch
}
```

Якщо в операторах блоку try помилок не виникло, то після їх виконання використовується блок finally (якщо він є). Після цього управління передається команді, що розташована після блоків try-catch-finally. Оператор finally є не обов'язковим, але досить корисним. Блок оператора finally виконується незалежно від того, згенеровано виняток чи ні. Якщо виняток згенеровано, то блок оператора finally виконується навіть і тоді, коли жоден з операторів catch не містить такого винятку.

Якщо у блоці try виникає помилка, а відповідний блок catch відсутній, то виняток викидається з методу. Він повинен опрацьовуватися зовнішнім відносно цього методу програмним кодом. Якщо ж у програмі виникла помилка, а опрацьовування цього типу помилки не передбачено, використовується обробник винятків за замовчуванням. Однак у такому разі програма завершить свою роботу.

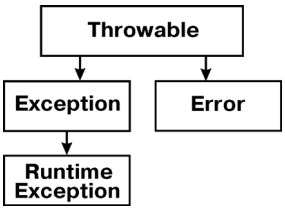

Рис. 1.1. Класи верхнього рівня ієрархії винятків

У мові Java для опрацьовування виняткових ситуацій передбачена ціла низка класів, а також підкласів. На верхньому їх рівні знаходиться суперклас Throwable (рис. 1.1), підкласи якого поділяються на дві гілки: Exception та Error.

Ліва гілка класів слугує для випадків, які повинен перехоплювати код прикладної програми. Під час створення власних типів винятків використовуються підкласи цієї гілки. Особливе значення має підклас RuntimeException, помилки для якого визначаються автоматично і перехоплюються у прикладних програмах користувача (ділення на нуль, вихід за межі вказаних розмірів масиву). Помилки типу Error виникають у самій виконавчій системі Java і пов'язані здебільшого з аварійними ситуаціями у системі.

На рис. 1.2 наведено програму, у якій демонструється варіант опрацювання виняткової ситуації у прикладній програмі на прикладі можливого виконання операції ділення на нуль з використанням блоку try-catch.

Isk_2.java ×

```
1  //Опрацювання винятку ділення на нуль
2  import java.util.Random;  //імпортування класу Random із пакету util
3  class Isk_2 {             //клас Isk_2 з методом main
4      public static void main(String args[]) {
5          int a=0, b;       //змінні цілочислового типу
6          Random x = new Random();       //об'єкт x класу Random
7          for (int i = 0; i < 7; i++) {  //виконати 7 циклів
8              try {                      //блок try
9                  b = x.nextInt(10); //змінна набуває випадкове значення
10                 a = 100 / b;       //ділення на випадкове число
11             } catch (ArithmeticException e) {  //блок catch
12                 //виведення повідомлення на екран
13                 System.out.println("увага, ділення на нуль"); //
14             }
15             System.out.println("на циклі №" + (i + 1) + (" a=") + a);
16         }
17     }}
```

Рис. 1.2. Програма опрацювання винятку ділення на нуль

Помилка ділення на нуль відноситься до класу ArithmeticException, який є підкласом класу RuntimeException. Для генерування випадкових чисел створюється об'єкт x класу Random за допомогою команди Random x=new Random (). Для того щоб цей клас був доступний, він імпортується за допомогою команди java.util.Random (клас Random належить пакету java.util). Випадкове ціле число в діапазоні від 0 до 10 генерується за допомогою методу nextInt (10), який викликається з об'єкта x. У круглих дужках цього методу вказується максимальне значення випадкового числа, що генерується. Нижня границя

випадкових чисел рівна нулю. У програмі генерується 7 випадкових чисел. Якщо будь-яке з них рівне нулю, виводиться повідомлення ділення на нуль, і змінна a набуває нульове значення. Після цього виконання циклів буде продовжено.

Один із варіантів виконання програми може бути таким:

```
"C:\Program ...
на циклі №1 a=100
на циклі №2 a=33
увага, ділення на нуль
на циклі №3 a=33
на циклі №4 a=50
на циклі №5 a=20
на циклі №6 a=14
на циклі №7 a=12

Process finished with exit code 0
```

Інколи доцільно виводити на екран опис самого винятку, що виник у процесі виконання програми. Для цього необхідно команду, що слідує безпосередньо за командою }catch (ArithmeticException e) { записати так: System.out.println ("увага, ділення на нуль"+e);. Для даного прикладу в цьому випадку на екран буде виведено: **увага, ділення на нуль java.lang.ArithmeticExcertion: / by zero.**

Якщо в наведеній програмі був би відсутній блок try-catch, то під час спроби ділення на нуль самою виконавчою системою Java буде згенеровано виняток, який буде перехоплено самою виконавчою системою, що врешті-решт призведе до зупинення програми й виведення на екран відповідного повідомлення. Вилучимо, наприклад, із програми, зображеної на рис. 1.2, блок try-catch. Програма набуде змісту, що зображений на рис. 1.3.

Isk_3.java ×

```
1  //Код без опрацювання винятків
2  import java.util.Random;  //імпортування класу Random із пакету util
3  class Isk_3 {             //клас Isk_3 з методом main
4      public static void main(String args[]) {
5          int a = 0, b = 0;          //змінні цілочислового типу
6          Random x = new Random();  //створення об'єкту x класу Random
7          for (int i = 0; i < 7; i++) {  //виконати 7 циклів
8              b = x.nextInt(10);     //змінна набуває випадкове значення
9              a = 100 / b;           //ділення на випадкове число
10             //виведення повідомлення на екран
11             System.out.println("на циклі №" + (i + 1) + (" a=") + a);
12         }
13     }}
```

Рис. 1.3. Програма без опрацювання виняткових ситуацій

Після виконання цієї програми може бути видано такий варіант результату:

```
"C:\Program ...
Exception in thread "main" java.lang.ArithmeticException: / by zero
на циклі №1 a=25
    at Isk_3.main(Isk_3.java:9)
на циклі №2 a=12
на циклі №3 a=11
на циклі №4 a=20 <5 internal calls>

Process finished with exit code 1
```

Тут видно, що на п'ятому циклі програма зупинила своє виконання, тому що на цьому кроці виникло ділення на нуль.

Зазначимо ще раз, що будь-який виняток, який не перехоплено прикладною програмою, обов'язково буде перехоплено й опрацьовано стандартним обробником системи. Цей обробник виводить на екран символьний рядок англійською мовою, який є описом винятку.

Вище використовувалися два класи винятків (ділення на нуль і вихід індексів за межі допустимих), які містяться в пакеті java.lang. Вони є винятками, які називаються такими, що не перевіряються, тому що компілятор не перевіряє, опрацьовуються або генеруються вони у якомусь методі. Такий тип винятків не обов'язково включати до списку оператора throws (з ним ми познайомимося пізніше) в оголошенні методу. У цьому пакеті є ще декілька класів, які є підкласами RuntimeException. Основні з них наведено у табл. 1.1.

Таблиця 1.1

Підкласи винятків, що не перевіряються, пакета java.lang

Виняток	Опис
ArithmeticException	Арифметична помилка, наприклад ділення на 0
ArrayIndexOutOfBoundsException	Індекс за межами допустимих
ArrayStoreException	Елементу масиву присвоюється об'єкт несумісного типу
ClassCastException	Неправильне перетворення типів
IllegalArgumentException	Недопустимий аргумент під час виклику методу
IllegalMonitorStateException	Недопустима контрольна операція (наприклад, очікування незаблокованого потоку виконання)