МЕТОДИ І ЗАСОБИ. ОБРОБКА ВИКЛЮЧЕНЬ

Лекція 6.1

доц. кафедри Інформатики
Сінельнікова Т.Ф.

ХНУРЕ, Кафедра Інформатики

informatika@kture.kharkov.ua

- Виключення
- Основні принципи обробки виключень
- Типи виключень
- Невідловлені винятки
- Використання операторів try і catch
- Множинні оператори catch
- Вкладені оператори try
- Оператор throw
- Методи з ключовим словом throws
- Блок finally
- Вбудовані виключення Java
- Створення власних підкласів виключень
- Використання виключень

Виключення

- Виключення - це аварійний стан, що виникає в кодової послідовності під час виконання.
- Іншими словами, виняток - це помилка часу виконання.
- В машинних мовах, не підтримують обробку винятків, помилки повинні бути перевірені і оброблені вручну - зазвичай за допомогою кодів помилки, і т. д.
- Обробка виключень у Java переносить управління обробкою помилок часу виконання в об'єктно-орієнтоване русло.

Основні принципи обробки виключень

- Виняток у мові Java - це об'єкт, який описує виключну (т. е. помилкову) ситуацію, що сталася в деякій частині коду.
- Коли виняткова ситуація виникає, створюється об'єкт, що представляє цей виняток, і «вкидається» в метод, що викликав помилку.

Основні принципи обробки виключень

- У свою чергу, метод може вибрати, обробляти чи виключення самому або передати його кудись ще.
- У кожному разі, в деякій точці виключення «захоплюється» і обробляється.

Основні принципи обробки виключень

- Винятки можуть генеруватися виконавчою системою Java, або ваш код може згенерувати їх "вручну".
- Викидаються винятки стосуються фундаментальних помилок, які порушують обмеження середовища виконання або правила мови Java.
- Винятки, згенеровані вручну, зазвичай використовуються, щоб повідомити викликає програмі про деяку аварійної ситуації.

Основні принципи обробки виключень

- Обробка виключень у Java управляється за допомогою п'яти ключових слів - try, catch, throw, throws і finally.
- Програмні оператори, які потрібно контролювати щодо винятків, містяться в блоці try.
- Якщо в блоці try відбувається виняток, кажуть, що воно викинуто (thrown) цим блоком.
- Ваш код може перехопити (catch) це виключення (використовуючи оператор catch) і обробити його деяким раціональним способом.

Основні принципи обробки виключень

- Винятки, що генеруються виконавчої (run-time) системою Java, викидаються автоматично.
- Для "ручного" викиду винятку використовується ключове слово throw. Будь-яке виключення, яке викинуто з методу, слід визначати за допомогою ключового слова throws, що розміщується в заголовному пропозиції визначення методу.
- Будь код, який обов'язково повинен бути виконаний перед поверненням з try-блоку, розміщується в finally-блоці, зазначеному в кінці блокової конструкції try {... }-catch {... }-finally {...}.

Основні принципи обробки виключень

- Загальна форма блоку обробки виключень:
  try {
  / / Блок коду для контролю над помилками} catch (ExceptionTypel exOb) {
  II обробник виключень для ExceptionTypel} catch (ExceptionType2 exOb) {
  II обробник виключень для ExceptionType2}
  II ... [Finally {
  / / Блок коду для обробки перед поверненням з try блоку
  }]
- Тут ExceptionType - тип виключення, яке виникло; ExOb - об'єкт цього виключення, Finally-блок - не обов'язковий.

Типи виключень

- Всі типи винятків є підкласами вбудованого класу Throwable. Throwable являє собою вершину ієрархії класів винятків. Безпосередньо нижче Throwable знаходяться два підкласи, які поділяють виключення на дві різні гілки.
- Одна гілка очолюється класом Exception. Цей клас використовується для виняткових станів, які повинні перехоплювати програми користувача. Це також клас, в підкласах якого ви будете створювати ваші власні замовні типи винятків.
- Іншу гілку очолює клас Error, що визначає виключення, перехоплення яких вашою програмою при нормальних обставин не очікується.
- Винятки типу Error застосовуються виконавчою системою Java для вказівки помилок, що мають відношення безпосередньо до середовища часу виконання.

Невідловлені винятки

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

- Коли виконавча система Java виявляє спробу поділу на нуль. вона створює новий об'єкт виключення і потім викидає його.

Java.lang.ArithmeticException: / by zero / / повідомлення про помилку? At ExcO, main (ExcO.Java: 4) / / траса стека

- ім'я класу (ExcO), ім'я методу (main), ім'я файлу (ExeO.java) і номер рядка (4).

Використання операторів try і catch

```
class Exc2 (
public static void main (String aros [j]) {
int d, a;
try {/ / контролювати блок коду
d = 0;
a = 42 / d;
System.out.println ("Це не буде надруковано.");}
catch (ArithmeticException e) {/ / перехопити помилку
/ / Ділення на нуль
System.out.println ("Поділ на нуль.");}
System.out.println ("Після оператора catch.");
}
```

- Ця програма генерує наступний висновок:

Поділ на нуль. Після оператора catch.
Звернення до println () всередині блоку try ніколи не виконується.

Множинні оператори catch

- В деяких випадках на одній ділянці коду може виникнути більше одного винятку. Після того як цей catch-оператор виконається, інші - обходяться, і виконання продовжується після блоку try / catch.

```
class MultiCatch {
    public static void main (String args [])
    {Try {
    int a = args. length;
    System.out.println ("a =" + a);
    int b = 42 / a; int c [] = {1}; c [42] = 99;}
    catch (ArithmeticException e) (
    System.out.println ("Поділ на нуль:" + e);}
      catch (ArrayIndexOutOfBoundsException e) {
    System.out.println ("Індекс елемента масиву oob:" + e);}
    System.out.println ("Після блоку try / catch.");}}
```

- Ця програма викине виняток "поділ на нуль", якщо вона буде запускатися без параметрів командного рядка.
- Якщо не виникло перше виключення, то це викличе виключення ArrayindexOutofBoundsException, так як цілочисельний масив з має довжину 1, тоді як програма намагається призначити деяке значення його сорок другого елементу з [42].

Вкладені оператори try

- Оператори try можуть бути вкладеними.
- Один try-оператор може знаходитися всередині блоку іншого оператора try.
- При вході в блок try контекст відповідного винятку поміщається в стек.
- Якщо внутрішній оператор try не має catch-обробника для специфічного винятку, стек розкручується, і проглядається наступний catch-обробник try-оператора.
- Процес триває до тих пір, поки не буде досягнутий відповідний catch-оператор, або поки всі вкладені оператори try не будуть вичерпані.
- Якщо узгоджується оператора catch немає, то виключення обробить виконавча система Java.

Вкладені оператори try

```
/ / Приклад вкладених try-операторів, class
    NestTry {
    public static void main (String args []) {
      try {
    int a = args.length;
    / * Якщо немає аргументів командного рядка,
    наступний оператор буде генерувати
    виключення поділу на нуль. * /
    int b = 42 / a;
    System.out.println ("a =" + a);
    try {/ / вкладений try-блок
    / * Якщо використовується один аргумент
    командного рядка, то наступний код буде
    генерувати виключення поділу на нуль. * /
    if (a == 1) a = a / (a-a); / / ділення на нуль
```

```
/ * Якщо використовується два аргументи
    командного рядка, то генерується
    виключення виходу за кордон масиву. * /
    if (a == 2) {
    int c [] = {1};
    c [42] = 99; / / генерувати виняток
    } / / Виходу за кордон масиву
    }
      catch (ArraylndexOutOfBoundsException e) {
    System.out.println ("Індекс виходить за кордон
    масиву:" + e);}
    }
    catch (ArithmeticException e) {
    System.out.println («Ділення на нуль:" + e);}}}
```

Оператор throw

- Ваша програма може сама явно викидати винятку, використовуючи оператор throw. Загальна форма оператора throw така:

Throw Throwableinstance;

- Тут Throwableinstance повинен бути об'єктом типу Throwable або підкласу Throwable. Прості типи, такі як int або char, а також не-Throwable-класи (типу string і object) не можуть використовуватися як виключення.
- Є два способи отримання Throwable-об'єкта: використання параметра в пропозиції catch або створення об'єкта за допомогою операції new.

Методи з ключовим словом throws

- Якщо метод здатний до породження виключення, яке він не обробляє, він повинен визначити свою поведінку так, щоб викликають методи могли самі охороняти себе від даного виключення.
- Це забезпечується включенням пропозиції throws в заголовок оголошення методу.
- Пропозиція throws перераховує типи винятків, які метод може викидати.
- Це необхідно для всіх винятків, крім винятків типу Error, RuntimeException або будь-яких їх підкласів.
- Всі інші винятки, які метод може викидати, повинні бути оголошені в реченні throws.

Методи з ключовим словом throws

- Якщо ця умова не дотримана, то відбудеться помилка часу компіляції.
- Загальна форма оголошення методу, яке включає пропозицію throws:

  type method-name (parameter-list) throw exception-list {
  / / Тіло методу}

- Тут exception-list - список розділених комами винятків, які метод може викидати.

Методи з ключовим словом throws

```
class ThrowsDemo  {
static void throwOne()
throws  IllegalAccessException   {
System.out.println("Внутри throwOne.");
throw new IllegalAccessException("demo");
}
public static void main(String args[])
{ try {
throwOne();
}
catch (IllegalAccessException e)
( System.out.println("Выброс " + e); } } }
```

Висновок, згенерований виконанням цієї програми:

Всередині throwOne.
Викид Java.lang.IllegalAccessException: demo

Блок finally

- Коли виняток викидається, виконання методу має досить нерівний, нелінійний шлях, який змінює нормальне проходження потоку через метод.
- В залежності від того, як кодований метод, виключення може викликати навіть передчасний вихід з нього.
- Наприклад, якщо метод відкриває файл для введення і закриває його для висновку, то ви навряд чи захочете, щоб закриває файл код був обійдений механізмом обробки винятків.
- Для реалізації цієї можливості і призначене ключове слово finally.

Блок finally

```
// Демонстрирує finally,
class FinallyDemo {
// Вихід з методу через виключення,
static void procAO { try {
System.out.println(«Усередині procA");
    throw new
    RuntimeException("demo"); }
finally {
System.out.println("finally для procA ");
    } }
// Повернення з try-блоку.
static void procBO { try {
System.out.println(«Усередині procB");
    return; }
finally {
System.out.println("finally для procB ");
    } }
// Нормальне виконання try-блоку,
static void procCO { try {
System.out.println(«Усередині procC");
    }
finally {
System.out.println("finally procC"); } }
public static void main(String args[]) {
try {
procA(); ) catch (Exception e) {
System.out.println(«Виключення
    викинуте"); }
procB(); procC(); } }
```

Вбудовані виключення Java

- Підкласи неконтрольованих винятків Java

Виключення	Значення
ArithmeticException	Арифметична помилка типу поділу на нуль
ArraylndexOutOfBoundsException	Індекс масиву знаходиться поза межами
ArrayStoreException	Призначення елементи масив несумісного типу
ClassCastException	Неприпустиме приведення типів
UlegalArgirnentException	При виклику методу використаний незаконний аргумент
IllegalMonitorStateException	Незаконна операція монітора, типу очікування на розблокованому потоці
UlegalStateException	Середа або додаток знаходяться в некоректному стані
UlegalThreadStateException	Необхідна операція не сумісна з поточним станом потоку
IndexOutOfBoundsException	Деякий тип індексу перебуває поза межами
NegativeArraySizeException	Масив створювався з негативним розміром
NullPointerException	Неприпустиме використання нульової посилання
NumberFormatException	Неприпустиме перетворення рядка в числовий формат
SecurityException	Спроба порушити захист
stringlndexOutOfBoundsException	Спроба індексувати поза межами рядка
OnsupportedOperationException	Зустрілася не підтримується операція

Вбудовані виключення Java

- Контрольовані винятки, визначені в java. lang

Виключення	Значення
ClassNotFoundException	Клас не знайдено
CloneNotSupportedException	Спроба клонувати об'єкт, який не реалізує інтерфейс Cloneable
IllegalAccessException	Доступ до класу відхилений
instantiationException	Спроба створювати об'єкт абстрактного класу або інтерфейсу
interruptedException	Один потік був перерваний іншим потоком
NoSuchFieldException	Необхідне поле не існує
NoSuchMethodException	Необхідний метод не існує

Створення власних підкласів виключень

- Клас Exception не визначає жодних власних методів, а успадковує ці методи від класу Throwable.
- Таким чином, всім винятків, навіть тим, що ви створюєте самі, доступні методи Throwable.
- Методи, визначені в Throwable

Метод	Опис
Throwable fillInStackTrace ()	Повертає Throwable-об'єкт, який містить повну трасу стека. Цей об'єкт може бути викинутий повторно
String getLocalizedMessage()	Повертає локалізоване опис виключення
String getMessage()	Повертає опис виключення
void printstackTrace()	Відображає трасу стека
void printStackTrace (Printstream stream)	Посилає трасу стека вказаною потоку
void printStackTrace (PrintWriterstream)	Посилає проекцію прямої стека вказаною потоку
string tostring()	Повертає string-об'єкт, що містить опис виключення. Цей метод викликається з println () при виведенні Throwable-об'єкта

Створення власних підкласів виключень

```
class MyException extends Exception {
    private int detail;
MyException(int a) {
detail = a;
public String toString() {
return "MyException[" + detail + "]"; } }

class ExceptionDemo {
static void compute(int a) throws
    MyException { System.out.printIn("
    Викликаний compute(" + a + "));
if(a > 10)
throw new MyException(a);
System.out.println("Нормальний
    вихід"); }
public static void main(String args[]) { try
    {
compute(1); compute(20); } catch
    (MyException e) {
System.out.printIn("Викинуто " + e); } }
    }
```

- Викликаний compute(1)
- Нормальний вихід
- Викликаний compute(20)
- Викинуто MyException[20]

Використання виключень

- Обробка винятків забезпечує потужний механізм управління комплексними програмами, що володіють безліччю динамічних характеристик часу виконання.
- Важливо представляти механізм try-throw-catch, як досить ясний спосіб обробки помилок і незвичайних граничних умов в логіці програми.
- Коли відбудеться відмова методу, нехай він сам викине виняток. Це більш ясний спосіб обробки режимів відмови.
- Операції обробки виключень Java не потрібно розглядати як загальний механізм для нелокального розгалуження.
- Якщо ви так зробите, це тільки заплутає ваш код і утруднить його підтримку.