Виключення (Exception)

Означення та сенс Exception у Java

Виключення — ***об`єкт***, який описує виключну(помилкову) ситуацію в коді програми. При кожній такий ситуації *в методі*, де вона виникла **створюється та передається об`єкт *класу Exception***.

Ситуації коли виникають виключення:

1) Виклик throw (власноруч створені)

2) Виключення було знайдено Jvm (правила виконання виразів або обмеження Jvm):

обчислення виразу — помилка (ділення на нуль)

помилка під час завантаження, лінковки та ініціалізації програми (викликається підклас LinkageError).

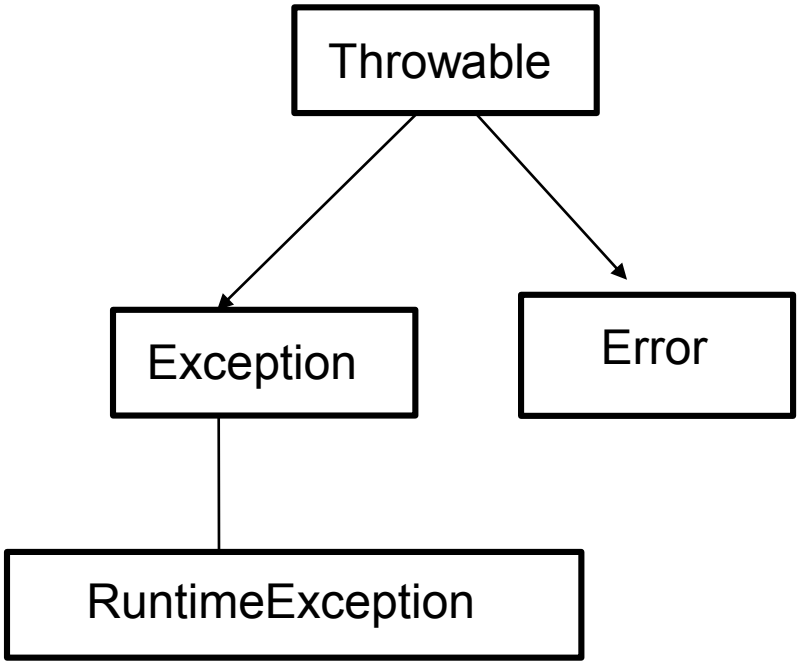
зовнішня помилка під час виділення ресурсів (VirtualMachineError).

Виключення в цих випадках виникає в тому місці, де зустрівся помилковий вираз.

3) Помилка сінхронізації

Базова ієрархія виключень

Всі виключення є нащадками суперкласу Throwable та його підкласів Error та Exception з пакету java.lang.



Виключення типу Error виникають тільки під час виконання програми. Такі виключення повязані з серйозними помилками, до прикладу, з переповненням стеку, не підлягають виправленню та не можуть оброблюваться додатком.

Класи винятків, успадкованих від класу RuntimeException

ArithmeticException - Арифметична помилка: поділ на нуль і ін.
ArrayIndexOutOfBoundsException - Індекс масиву знаходиться поза межами
ArrayStoreException - Призначення елементу масиву несумісного типу
ClassCastException - Неприпустиме приведення типів
ConcurrentModificationException - Некоректна модифікація колекції
IllegalArgumentException - При виклику методу використаний незаконний аргумент
IllegalMonitorStateException - Незаконна операція на розблокованому екземплярі
IllegalStateException - середовище або програми містяться в некоректному стані
IllegalThreadStateException - операція не сумісна з поточним станом потоку
IndexOutOfBoundsException- Деякий тип індексу перебуває поза межами
NegativeArraySizeException - Масив створювався з негативним розміром
NullPointerException - Неприпустиме використання нульового посилання
NumberFormatException - Неприпустиме перетворення рядка в числовий формат
StringIndexOutOfBoundsException - Спроба індексації поза межами рядка
UnsupportedOperationException - Зустрілася операція, що не підтримується

Класи винятків, не успадкованих від класу RuntimeException

ClassNotFoundException - клас не найден;
CloneNotSupportedException - спроба клонувати об'єкт з класу, який не реалізує інтерфейс Cloneable;
IllegalAccessException - заборонений доступ до класу;
InstantiationException - спроба створити об'єкт абстрактного класу або інтерфейсу;
InterruptedException - один потік виконання перерваний іншим потоком;
NoSuchFieldException - запитувана поле не існує;
NoSuchMethodException - запитуваний метод не існує;
ReflectiveOperationException - суперклас винятків, пов'язаних з рефлексією.

Обробка Виключень: try/catch/finally

```
try{
     // тут якийсь програмний код, що може викликати виключення
     //… або декілька виключень
}
//обробка першого виключення
catch(<Тип Виключення 1> ex_obj1){
   // код що потрібен для обробки першого виключення
}
 // обробка другого виключення (якщо воно є)
catch(<Тип Виключення 2> ex_obj2){
   // код що потрібен для обробки другого виключення
}
****
/* якщо нам потрібен код, що виконується у незалежно від того булу помилка чи ні */
finally{
  // блок кода що викнується після try
}
```

Ділення на нуль: без обробки

```
class Exc0 {
  public static void main(String args[]) {
    int d = 0;
    int a = 42 / d;
  }
}
/* При виконанні цього коду:
Java.lang.ArithmeticException: / by zero at Exc0.main (Exc0.java:5) */
class Exc1 {
  static void subroutine() {
    int d = 0;
    int a = 10 / d;
  }
  public static void main(String args[]) {
    Exc1.subroutine();
  }
}
/*Java.lang.ArithmeticException: / by zero at  Exc1.subroutine(Exc1.java:5)
at Exc1.main (Exc1.java:8)*/
```

Ділення на нуль: з обробкою

```
class Exc2 {
  public static void main(String args[]) {
    int d, a;

    try { // оброблюємо виключення в блоці коду
      d = 0;
      a = 42 / d;
      System.out.println("This will not be printed.");
    } catch (ArithmeticException e) { // ловимо (catch) divide-by-zero error
      System.out.println("Division by zero.");
    }
    System.out.println("After catch statement.");
  }
}
/* Результат: Division by zero.
 After catch statement. */
```

Обробка декількох виключень

```
// Відловлювання декілька catch
class MultipleCatches {
  public static void main(String args[]) {
    try {
      int a = args.length;
      System.out.println("a = " + a);
      int b = 42 / a; // помилка якщо args  - порожній
      int c[] = { 1 };
      c[42] = 99; // помилка якщо довжина args менше 43
    } catch(ArithmeticException e) {
      System.out.println("Divide by 0: " + e);
    } catch(ArrayIndexOutOfBoundsException e) {
      System.out.println("Array index oob: " + e);
    }
    System.out.println("After try/catch blocks.");
  }
}
```

Ділення на нуль: з обробкою

```
// Оброблюємо помилку та продовжуємо роботу
import java.util.Random;
class HandleError {
  public static void main(String args[]) {
    int a=0, b=0, c=0;
    Random r = new Random();

    for(int i=0; i<32000; i++) {
      try {
        b = r.nextInt();
        c = r.nextInt();
        a = 12345 / (b/c);
      } catch (ArithmeticException e) {
        System.out.println("Exception: " + e); /* Exception:
                                                  java.lang.ArithmeticException: / by zero */
        a = 0; // присвоюємо відповідь 0 та працюємо далі
      }
      System.out.println("a: " + a);
    }
  }
}
```

Обробка декількох виключень

```
// Відловлювання декілька catch
class MultipleCatches {
  public static void main(String args[]) {
    try {
      int a = args.length;
      System.out.println("a = " + a);
      int b = 42 / a; // помилка якщо args  - порожній
      int c[] = { 1 };
      c[42] = 99; // помилка якщо довжина args менше 43
    } catch(ArithmeticException e) {
      System.out.println("Divide by 0: " + e);
    } catch(ArrayIndexOutOfBoundsException e) {
      System.out.println("Array index oob: " + e);
    }
    System.out.println("After try/catch blocks.");
  }
}
```

Код, що не досягається

```
/* Підклас не може бути пойманим після того
як є спійманим суперклас. «Мертвий код» з
точки зору Java -помилка */
class SuperSubCatch {
  public static void main(String args[]) {
    try {
      int a = 0;
      int b = 42 / a;
    } catch(Exception e) {
      System.out.println("Generic Exception
catch.");
    }
    /* Цей catch недосяжний, бо
ArithmeticException - підклас Exception. */
    catch(ArithmeticException e) {
 // ERROR - unreachable
      System.out.println("This is never reached.");
    }
  }
}
```

```
class SuperSubCatch {
  public static void main(String args[]) {
    try {
      int a = 0;
      int b = 42 / a;
    } catch(Exception e) {
      System.out.println("Generic
Exception catch.");
    }
 // тут все ОК
}catch(ArithmeticException e) {
      System.out.println("This is reached.");
    }
 catch(Exception e) {
      System.out.println("Generic
Exception catch.");
    }
```

Виведення повного шляху помилки та однакова обробка виключень

```
try {

// деякий блок операцій
} catch(NumberFormatException e) {

e.printStackTrace();
} catch(ClassNotFoundException e) {

e.printStackTrace();
} catch(InstantiationException e) {

e.printStackTrace();
}
```

```
try {

// деякий блок операцій
} catch(NumberFormatException |
ClassNotFoundException |
InstantiationException e) {

e.printStackTrace();
}
```

Вкладені блоки виключення

```
class NestTry {
  public static void main(String args[]) {
    try {
      int a = args.length;  /* при відсутності аргументів буде виключення */
      int b = 42 / a;
      System.out.println("a = " + a);
      try { // вкладений блок try
        /*  якщо 1 аргумент — буде ділення на 0 */
       if(a==1) a = a/(a-a); // division by zero
        /* якщо 2 аргументи: out-of-bounds exception. */
       if(a==2) {
         int c[] = { 1 };
         c[42] = 99; // out-of-bounds exception
       }
     } catch(ArrayIndexOutOfBoundsException e) {
       System.out.println("Array index out-of-bounds: " + e);
     }
    } catch(ArithmeticException e) {
     System.out.println("Divide by 0: " + e);
    }
  }
}
```

Вкладені блоки виключення

```
/* Try можуть бути вкладені у послідовні виклики методів */
class MethNestTry {
  static void nesttry(int a) {
    try { // вкладений try block
       if(a==1) a = a/(a-a); // division by zero
       if(a==2) {
           int c[] = { 1 };
           c[42] = 99; // генерує out-of-bounds exception
      }
    } catch(ArrayIndexOutOfBoundsException e) {
      System.out.println("Array index out-of-bounds: " + e);
    } }
  public static void main(String args[]) {
    try {
      int a = args.length;
      int b = 42 / a;      /* divide-by-zero exception. */
      System.out.println("a = " + a);
      nesttry(a);
    } catch(ArithmeticException e) {
      System.out.println("Divide by 0: " + e);
    } }}
```

Оператор throw

Виключення викликається оператором throw:
1) Якщо обєкт потрібного класу інсує:

throw обєктThrowable;

```
Exception e = new ArithemticException();
****
```

throw e;

2) Якщо його потрібно створити за допомогою оператора new:
throw new КласВиключення(параметри конструктору)

```
throw new IilegalArgumentException();
```

```
// Демострація throw.
class ThrowDemo {
  static void demoproc() {
    try {
      throw new NullPointerException("demo");
    } catch(NullPointerException e) {
      System.out.println("Caught inside demoproc.");
      throw e; // повторне кидання виключення (re-throw the exception)
    }
  }

  public static void main(String args[]) {
    try {
      demoproc();
    } catch(NullPointerException e) {
      System.out.println("Recaught: " + e);
    }
  }
}
```

Оператор throws

```
ТипМетоду НазваМетоду(параметри)  throws Виключення1, Виключення2,...
{
    // тіло методу
    // виклик throw або методу, що викликає ці виключення
}
```

```
// Без throws - помилка
class ThrowsDemo {
  static void throwOne() {
    System.out.println("Inside throwOne.");
    throw new
IllegalAccessException("demo");
  }
  public static void main(String args[]) {
    throwOne();
  }
}
```

```
// тепер - коректно
class ThrowsDemo {
  static void throwOne() throws
IllegalAccessException {
    System.out.println("Inside throwOne.");
    throw new IllegalAccessException("demo");
  }
  public static void main(String args[]) {
    try {
      throwOne();
    } catch (IllegalAccessException e) {
      System.out.println("Caught " + e);
    }
  }
}
```

Оператор throws

```
public class IllegalResourceException
extends Exception {}

public static void
loadResource(SameResource f) throws
IllegalResourceException {
if (f == null || !f.exists() || !f.isCreate()) {
throw new IllegalResourceException();
}
// море коду ( more code)
}
```

```
package b1.conn;
public class Connector {
public static void
loadResource(SameResource f) {
if (f == null || !f.exists() || !f.isCreate()) {
/* генерация исключения */
throw new IllegalArgumentException();
// или собственное,
 throw new IllegalResourceException();
}
// море коду ( more code)
}
}
```

Оператор throws

```
package b1.conn;
public class Runner {
public static void main(String[ ] args) {
SameResource f = new SameResource(); //
SameResource f = null;
try{
/*необовязковий лише при гарантованій
коректності значень параметрів */
  Connector.loadResource(f);
} catch(IllegalArgumentException e) {
   System.err.print("обработка unchecked-
виключения з методу: " + e);
}
}
```

```
package b1.conn;
public class SameResource {
// поля, конструкторы
public boolean isCreate() {
// more code
}
public boolean exists() {
// more code
}
public void execute() {
// more code
}
public void close() {
// more code
}
}
```

Створення виключень

```
// Ця прграма створює а власний тип
виключення.
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.println("Called compute(" + a
+ ")");
    if(a > 10)
      throw new MyException(a);
    System.out.println("Normal exit");
  }

public static void main(String args[]) {
    try {
      compute(1);
      compute(20);
    } catch (MyException e) {
      System.out.println("Caught " + e);
    }
  }
}
```

Методи класу Throwable

Конструктори
Throwable() створює throwable - null
Throwable(String message) створює throwable з даним повідмленням.
Throwable(String message, Throwable cause)Новий throwable з повідомленням та причиною.
protected Throwable(String message, Throwable cause, boolean enableSuppression, boolean writableStackTrace)
Throwable(Throwable cause) створює виключення з причиною(cause==null ? null : cause.toString())

void addSuppressed(Throwable exception) — додає до подавлених виключень
Throwable fillInStackTrace() Заповнює трасування стеку виконання
Throwable getCause() повертає причину Throwable або null якщо її нема або вона невідома
String getLocalizedMessage() створює локалізований опис Throwable.
String getMessage() повертає повідомлення про Throwable.

Методи класу Throwable

StackTraceElement[] getStackTrace() програмований доступ до списку, що повертає printStackTrace()
Throwable[] getSuppressed() повертає масив всіх подавлених виключень з try-операторами з ресурсами
Throwable initCause(Throwable cause) Ініціалізує *причину* цього throwable при даному значенні.
String printStackTrace() друкує throwable до стандартного потоку повдомлень про помилки
String printStackTrace(PrintStream s)друкує throwable у вказний поток друку.
String printStackTrace(PrintWriter s) друкує throwable у вказний поток запису.
void setStackTrace(StackTraceElement[] stackTrace) встановлює елементи стека, для поверення getStackTrace() і друку printStackTrace()
String toString() повертає опис throwable.

Створення ланцюгу виключень

```
 // Ланцюг виключень
class ChainExcDemo {
  static void demoproc() {
    // створити виключення
    NullPointerException e = new NullPointerException("top layer");
    // додаємо причину cause
    e.initCause(new ArithmeticException("cause"));
    throw e;
  }
  public static void main(String args[]) {
    try {
      demoproc();
    } catch(NullPointerException e) {
      // показуємо перше виключення -  top level exception
      System.out.println("Caught: " + e);
      // показуємо причину -  cause exception
      System.out.println("Original cause: " + e.getCause());
    }
  }
}
```

Створення виключень - 2

```
public class Coin {
private double diameter;
private double weight;
public double getDiameter() {
	return diameter;
}
public void setDiameter(double value) throws CoinLogicException {
	if(value <= 0) {
		throw new CoinLogicException("diameter is incorrect");
	}
diameter = value;
}
public double getWeight() {
	return weight;
}
public void setWeight(double value) {
weight = value;
}
}
```

Створення виключень - 2

```
public class CoinLogicException extends Exception {
public CoinLogicException() {
}
public CoinLogicException(String message, Throwable exception) {
      super(message, exception);     }
public CoinLogicException(String message) {
      super(message); }
public CoinLogicException (Throwable exception) {
      super(exception);}
}
****
public void doAction(String value) throws CoinTechnicalException {
      Coin ob = new Coin();
      try {
            double d = Double.parseDouble(value);
            ob.setDiameter(d);
      } catch (NumberFormatException e) {
            throw new CoinTechnicalException("incorrect symbol in string", e);
      } catch (CoinLogicException e) {
            System.err.println(e.getCause());
      }
}
```

Створення виключень-3

```
public class CoinTechnicalException extends Exception {
public CoinTechnicalException() {    }
      public CoinTechnicalException(String message, Throwable cause) {
       super(message, cause);      }
public CoinTechnicalException(String message) {
       super(message);     }
public CoinTechnicalException(Throwable cause) {
       super(cause);
}
} ****
public void doAction(String value) throws CoinLogicException {
       Coin ob = new Coin();
       try {
              double d = Double.parseDouble(value);
              ob.setDiameter(d);
       } catch (CoinException e) {
              throw e;
       }
}
```

```
// власне виключення - наслідник Exception
class MyException extends Exception{
  // перевантажена getLocalizedMessage()
  public String getLocalizedMessage()  {
    return "MyException.getLocalizedMessage()";
  }
}
public class Train {
    public static void main(String[] args) { /* Ввести число x. Якщо воно за межами [0..100],то
згенерувати MyException */
    int x;
    Scanner sc = new Scanner(System.in);
    System.out.print("x = ");
    x = sc.nextInt(); // ввести x
    try {// ловимо виключення
      if ((x<0)||(x>100)) throw new MyException();
      System.out.println("OK!");
    }
```

```
catch(MyException e)    {  // обробка MyException, демонстрація методів класу Throwable
    System.out.println("Return from getLocalizedMessage(): " + e.getLocalizedMessage());
    System.out.println("Return from getMessage(): " + e.getMessage());
    System.out.println("Method printStackTrace(): ");
    e.printStackTrace();
    System.out.println("Method toString(): " + e.toString());
    System.out.println("----------------------");
    System.out.println("Method getStackTrace(). Stack trace: ");
    StackTraceElement[] stE;
    stE = e.getStackTrace(); // метод getStackTrace()
    for (int i=0;i<stE.length;i++)
      System.out.println(stE[i].toString());
    System.out.println("----------------------");
    System.out.println("Method fillStackTrace(). Stack trace: ");
    Throwable tA = e.fillInStackTrace();
    StackTraceElement[] stE2 = tA.getStackTrace();
    for (int i=0; i<stE2.length; i++)
      System.out.println(stE[i].toString());
    System.out.println("----------------------");
  }
}
```

```
x = 200
Return from getLocalizedMessage(): MyException.getLocalizedMessage()
Return from getMessage(): null
Method printStackTrace():
Method toString(): MyException: MyException.getLocalizedMessage()
----------------------
Method getStackTrace(). Stack trace:
Train.main(Train04.java:36)
----------------------
Method fillStackTrace(). Stack trace:
Train.main(Train04.java:36)
----------------------
MyException: MyException.getLocalizedMessage()
```

Правила наслідування

1. Перевизначений метод в підкласі не може містити в інструкції throws винятків, що обробляються в відповідному методі суперкласу;

2. Конструктор підкласу повинен включити в свій блок throws всі класи виключень або їх суперкласу з блоку throws конструктора суперкласу, до якого він звертається при створенні об'єкта

```
// початковий класс
public class Stone {
public void build(String data) throws
ParseException {
      ***/* реалізация */
}
}
public class WhiteStone extends Stone {
// раніш створений клас
@Override
public void build(String data) {
      /* реалізация */
      System.out.println("Білий
                  кам`яний шар");
}
}
```

```
// клас дії
public class StoneAction {
public void buildHouse(Stone stone) {
      try {
      stone.build("some info");
      // обробка ParseException з підкласами
      } catch(ParseException e) {
            System.err.print(e);
      }
}}

public class BlackStone extends Stone {
@Override
public void build(String data) throws Exception {
// помилка компіляції
      System.out.println("чорний
кам`яний шар");
      /* реалізація*/
}}
```

Виключення assert

- **assert БулевийВираз : ВиразРядок**;
- **assert БулевийВираз;**

Вираз БулевийВираз може має значення типу boolean або Boolean,
Вираз ВиразРядок вираз, що повертає рядок(String). Якщо значення false, то гене-рується AssertionError - на консоль значения виразу expression (якщо воно є).

Приклад:

```
int age = ob.getAge();
assert (age >= 0): "NEGATIVE AGE!!!";
// далі щось кодимо ….
```

```
enum Mono { WHITE, BLACK };
String str = "WHITE"; // або "GRAY" встановлюємо колір
Mono mono = Mono.valueOf(str); // і в перерахуванні також
// … якийсь код ще
switch (mono) {

case WHITE : // щось кодимо
      *** break;

case BLACK : // ще щось кодимо
           ***
      break;

default :
      assert false : "Colored!";
}
// Для підключення/відключення ассертів в комендному рядку:
//java -enableassertions RunnerClass   (java -disableassertions RunnerClass) або
// java -ea RunnerClass  (java -da RunnerClass)
```