Тема 11 - Exceptions

20 юни 2018 г. 22:29

Exception е извън нормално състояние, което настъпва по време на изпълнение на програмата(at run time). С други думи exception е runtime грешка. Ако не разполагахме с тях щяхме да трябва да проверяваме за грешки ръчно и да връщаме error кодове.
Следният метод разглежда този подход:

```
int withdraw(int amount) {
    if (amount > balance) {
         return -1;
    }
    else {
         balance -= amount;
         return 0;
    }
}
```

Този метод се използва за изтегляне на някаква сума от сметка. Той използва традиционният начин за връщане на специална стойност в случая -1, когато възникне грешка. Главният проблем при този подход ,е че този, който вика метода трябва да е запознат, че той връща някакъв код за грешки и какъв е той, а в някои случай може и да има различни кодове за грешка, и трябва да обслужи грешката. Извикването ще изглежда така:

```
If(withdraw(200) == 0) {
    System.out.println("Parite sa istegleni");
} else {
    System.out.println("Vuznikna greshka");
}
```

Този подход се избягва чрез използването на exceptions:

```
void withdraw(int amount) throws BalanceException {
    if(amount > balance) {
         throw new BalanceException();
    }
    balance -= amount;
}
```

Самите exception-и представляват обект, както се вижда в случая, когато настъпи грешка се създава нов обект от тип BalanceException и се "хвърля". Грешки могат да се хвърлят от всякакъв метод дори конструктор.
В Java за управлението на exception-и се използват 5 запазени думи:

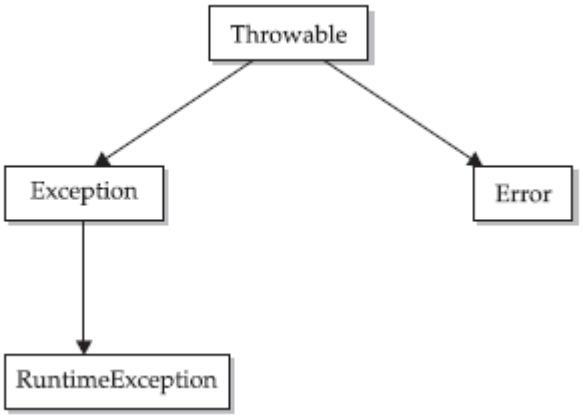
- **try** - определя блок от код, който искате да контролирате за exceptions
- **catch** - "улавя" exceptions и ги обслужва по някакъв зададен от вас начин
- **throw** - ръчно да "хвърляне" на exception (системно генерираните exceptions ги хвърля виртуалната машина на Java)
- **throws** - обявява какви грешки може да хвърли даден метод
- **finally** - ако има код, който независимо от това дали има грешка трябва да се изпълни се слага в finally блок.

Едно примерно използване на системата за управление на грешки:

```
try {
// блок от код, в който наблюдавате за грешки
}
```

```
catch (ExceptionType1 exOb) {
// обработване в случай че настъпи грешка от тип ExceptionType1
}
catch (ExceptionType2 exOb) {
// обработване в случай че настъпи грешка от тип ExceptionType2
}
// ...
finally {
// блок от код, който се изпълнява винаги в случай, че настъпи грешка
след изпълняването на try блока
}
```

Йерархията на грешки:



Throwable е класа, който оказва ,това че можете да хвърлите определена грешка.
Error класа представлява грешки, които вие не може да обслужите или да хващате, това обикновено са грешки свързани с средата за изпълнение на програмата(Stack overflow).
Exception е главният клас на обслужваните от вас грешки именно него може да наследите за да създадете собствени грешки, както беше BalanceException.
RuntimeException е важен под-клас от грешки, които са ви автоматично генерирани от програмата и това са грешки като деление на нула или грешен индекс при масивите.

Първо да погледнем какво става ако не обслужим грешките:

```
//Файл:Exc1.java
class Exc1 {
    static void subroutine() {
        int d = 0;
        int a = 10 / d;
    }
    public static void main(String args[]) {
        subroutine();
    }
}
Конзолата:
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Exc1.subroutine(Exc1.java:4)
        at Exc1.main(Exc1.java:7)
```

Java автоматично изкарва на конзолата нещо наречено callstack, което е просто последователността от извиквания, която е довела до грешката. Виждат се редовете ,в които е настъпила грешката. Това е полезно, когато дебъгвате някоя програма, но в

повечето случай бихте искали да обслужите грешката сами:

```
//Файл:Exc2.java
class Exc2 {
     static void subroutine() {
          int d = 0;
          int a = 10 / d;
     }
     public static void main(String args[]) {
          try{
               subroutine();
               System.out.println("Tozi red nqma da se izpylni");
          } catch(ArithmeticException exc) {
               System.out.println("Delenie na nula");
          }
     }
}
```

В случая просто извеждаме съобщение за грешката, но в други ситуации може да има сложни процедури ,през които се минава за да се запаси стабилно състоянието на програмата. Реда точно под извикването на метода subroutine() няма да се изпълни защото възникне ли грешка в try/catch блок хода на програмата веднага се предава на catch блока. Чрез използване на finally блок задаваме област, която ще се изпълни винаги независимо дали е постъпила грешка или не.

```
//Файл:Exc3.java
class Exc3 {
	static void subroutine() {
		int d = 0;
		int a = 10 / d;
	}
	public static void main(String args[]) {
		try{
			subroutine();
			System.out.println("Tozi red nqma da se izpylni");
		} catch(ArithmeticException exc) {
			System.out.println("Delenie na nula");
		} finally {
			System.out.println("Tozi red shte se izpylni vinagi");
		}
	}
}
```

В класа Throwable има няколко полезни метода, които служат за информация относно грешките.

```
//Файл:Exc5.java
class Exc4 {
	public static void main(String args[]) {
		int [] arr = new int[4];
		try {
			arr[4] = 5;
		} catch(IndexOutOfBoundsException exc) {
			System.out.println("Ime na greshkata:" + exc);
			System.out.println("Callstack:" + exc.getStackTrace());
			System.out.println("----------Izpolzvane na printStackTrace----------");
			exc.printStackTrace();
		}
	}
}
Конзолата:
Ime na greshkata:java.lang.ArrayIndexOutOfBoundsException: 4
Callstack:[Ljava.lang.StackTraceElement;@15db9742
----------Izpolzvane na printStackTrace----------
java.lang.ArrayIndexOutOfBoundsException: 4
	at Exc4.main(Exc4.java:9)
```

Първият метод е toString() (той се използва автоматично, когато се сложи обекта в поле за String), той ни дава името на грешката.
Вторият метод е getStackTrace(), той ни дава обект от тип StackTraceElement [] (масив, на конзолата се вижда неговият хеш код).
Третият метод се използва най-често за дебъгване, той принтира callstack-а или трейсва стека.

Както споменахме по-рано, ние можем да си създаваме собствени exceptions, като наследяваме класа Exception:

```
//Файл:ExceptionDemo.java
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 { // обявяваме, че
    този метод може да хвърли грешка от тип MyException
        System.out.println("Called compute(" + a + ")");
        if(a > 10) {
            throw new MyException(a);//хвърляме грешка и подаваме
            аргумент
        }
        System.out.println("Normal exit");
    }
    public static void main(String args[]) {
    try {
        compute(1);
        compute(20);
    } catch (MyException e) {
        System.out.println("Caught " + e);
    }
    }
}
```

В Java има голям набор от готови exceptions, които може да използвате да хвърляте от вашата програма, но когато имате нужда от ваша собствена наследявате класа Exception, и няма нужда да имплементирате никой метод или да го пренаписвате. Идеята зад създаване на нови грешки е в тяхното име и евентуално в някой, нов конструктор както е в примера или пренаписване на toString() метода за да включва някакъв детайл, който искате.

Задача:

//Файл:Point.java, Файл: Rectangle.java, Файл: ExceptionTesting.java, Файл: NegativePointException.java, Файл:WrongOrderOfPointsException.java

Вземете класовете от задача 2 от темата за масиви и този път не предполагаме нищо за входовете на конструкторите Point и Rectangle, но искаме да имат следните правила:
Point:
Ако точка е създадена с отрицателни координати(дори само един да е отрицателен), точката се създава с два нулеви координата. - използвайте хвърляне и хващане на exception от клас **NegativePointException** клас(който вие трябва да създадете).
Rectangle:
Ако някоя от точките е нулева да се хвърля **ZeroPointException**("Left Top"), където Left Top е точката, която е нулева(които и от двата координата да е нула).(Програмата ще открива първата нулева в поредността Left Top, Right Top, Left Bot, Right Bot)

Тествайте програмата, в отделен клас с име ExceptionTesting с main метод. Опитайте се да изредите всички вероятности.

Допълнителни линкове към темата:
https://docs.oracle.com/javase/tutorial/essential/exceptions/
https://docs.oracle.com/javase/7/docs/api/java/lang/Exception.html

Автор: Димитър Томов - xdtomov@gmail.com