

Исключения, проверки и дневници в Java. Дебъгване.

Лектор:

инж. Божидар Бацов

Грешки в приложенията

- Програмни грешки – възникват по вина на програмиста
- Потребителски грешки – възникват по вина на протребителя
 - Неподходящи входни данни
 - Неправилна конфигурация
- Системни грешки – липса на оперативна памет, файлови дескриптори и т.н.

Класически подход за следене за грешки

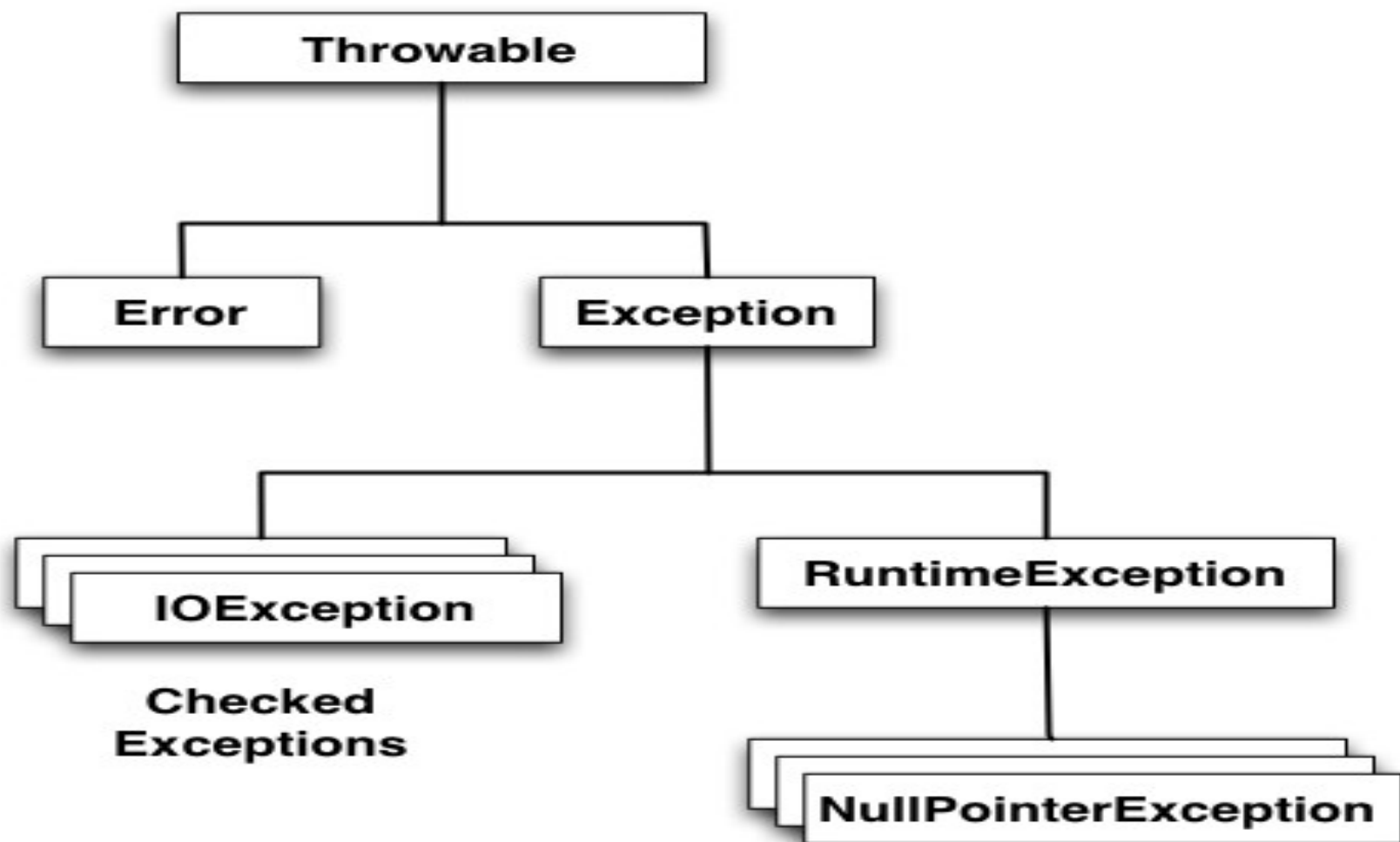
- Връщане на специални стойности от методи
 - -1
 - Null
- Проблеми на класическия подход
 - Върнатите стойности могат лесно да бъдат пренебрегнати
 - Не винаги има подходящи стойности

Пример в NetBeans

Исключение

- Специален обект, който сигнализира за случването на някое изключително(неочаквано) събитие
- Капсулира в себе си съобщение за грешка, stack trace
- Имаме инфраструктура за обработка на изключения

Йерархия на изключенията



Основни класове в йерархията

- Throwable – всички изключения го разширяват
- Error – базов клас за изключения свързани с грешки във виртуалната машина(не трябва да го разширявате)
- RuntimeException – базов клас за изключение свързани с програмни грешки
- Exception – всичко останало

Видове изключения

- Checked exceptions – изключения, чията обработка компилатора проверява автоматично(за това се казват checked)
- Unchecked exceptions – изключения, които възникват от програмни грешки(ArrayIndexOutOfBoundsException) и от грешки възникнали във виртуалната машина(OutOfMemoryError)

Исключения от стандартната библиотека

- Unchecked
 - IllegalArgumentException
 - IllegalStateException
 - ArithmeticException
 - NullPointerException
- Checked
 - FileNotFoundException

Дефиниране на нови ИЗКЛЮЧЕНИЯ

- Желателно да се използват максимално стандартните изключения, вместо да се създават нови
- За базови класове на новите изключения се използват `Exception` (за checked exceptions) и `RuntimeException`
- Не трябва да се разширяват класовете `Error` и `Throwable` директно

```
class MyException extends Exception {  
    public MyException(String message) {  
        super(message);    }  
    public MyException() {  
    public MyException(String message,  
        Throwable cause) {  
        super(message, cause);    }  
    }  
}
```

Работа с изключения

- Изключенията се сигнализируют посредством ключовата дума “throw”
 - `throw new SomeException();`
- Изключенията се прихващат в `try` блок
- Изключенията могат да бъдат препредадени към извикващия метод посредством `throws` директивата

Работа с исключения

```
public void someMethod() throws  
SomeException {...}
```

```
try {  
... }
```

```
catch (SomeException e) {
```

```
} finally {
```

```
}
```

Работа с изключения

- Има смисъл да се прихващат само изключения, за които можем да направим нещо
- Има смисъл да се пропагират само `checked` изключения
- Код в блок `finally` винаги се изпълнява – независимо дали е възниквано изключение или не

Stack trace

- Изключението съдържа в себе си информация известна като stack trace, която ни показва реда на извиквания на методи, довели до възникването на изключението
- Изключително полезен при дебъгване за определяна на причината за възникване на проблем

Опаковане на изключения

- Едно изключение може да е да причинено от друго
- Причината(cause) се задава в конструктора на новото изключение
- Мощна техника – обикновено се използва за опаковане на checked в unchecked изключения

Добри практики при работа с ИЗКЛЮЧЕНИЯ

- Предпочитайте стандартните изключения
- Използвайте ги само, когато наистина има нужда – обикновено възникването на непредвидима ситуация
- Не ги потискайте – това означава обработката им да бъде просто празен catch блок

Работа с проверки(assertations)

- Проверката се използва да верифициране коректността на стойности в определени места от програмата.
- Проверките се извършват с ключовата дума `assert`
- Проверките се активират на ниво виртуална машина с флага “-ea”

Водене на програмен журнал

- Журнал – информативни съобщения, чрез които можем да анализираме работата на дадено приложение
- Обикновено журналните съобщения се разделят на категории, в зависимост от тяхната важност – debug, info, warning, error и т.н.

Работа с проверки

- Ако проверката пропадне – програмата завършва с `AssertionError`
- `assert condition` – проверява се условието и ако не е изпълнено се генерира грешка без описание
- `assert condition : expression` – ако се генерира грешка вторият израз се използва за нейно описание

Смисъл на проверките

- Във фазата на разработка те се държат включени за да могат критични грешки да бъдат откривани моментално
- Във фазата на пускане(production) на приложението, те просто се изключват на ниво виртуална машина, няма нужда от промени в кода

Пример в NetBeans

Водене на програмен журнал

- Могат да се показват само в конзолата, да се съхраняват във файл или и двете
- Трябва да се имат в предвид и някакви ротиращи техниките за тях – размера им може да стане огромен в противен случай

Журнални библиотеки в Java

- Java Logging – част от JDK
- Log4j – най-популярната лог библиотека
- Simple Log
- Commons Logging, SLF4J – фасадни библиотеки, които вътрешно ползват една от гореизброените

Журнални библиотеки в Java

- Обикновено работа им се конфигурира посредством конфигурационен файл или програматично
- Могат да се задават параметри като нивото на съобщенията, които да се включват, тяхната описателност (verbosity), медиума, на който те да отиват (файл, екран) и т.н.

Пример в NetBeans

Добри практики при водене на журнал

- Всеки клас, който върши нещо трябва да има `Logger(private static final)`
- Журналните съобщения не трябва да се криптични, а ясни и описателни
- Дръжте конфигурацията си в конфигурационни файлове, а не в кода
- Предпочитайте фасадни библиотеки – позволяват да смените `underlying` импл.

Дебъгване на приложения с дебъгер

- Най-могъщата техника за търсене на грешки в приложения
- Дебъггера позволява изпълнението на програма да бъде прекъснато в определена точка(break point), да се следят стойностите на всички променливи, да се изпълняват твърдения едно по едно и т.н.

Работа с дебъгер

- Интегриранията ви среда може да работи директно с дебъгера – огромно улеснение
- Поставяте break points(line, method, etc)
- Стартирате приложението в debug режим
- Програмата спира(паузира) на първия срещнат break point

Пример в NetBeans

Добри практики при ползване на дебъгер

- Ако имате грешка, с която да работите – анализирайте стека от нея, за да може да изберете по-добро място за break points
- Развийте навик да преглеждате състоянието на всички по-важни стойности
- Избягвайте употреба на `method br.points`