СБЕРБАНК ТЕХНОЛОГИИ

ОБРАБОТКА ИСКЛЮЧЕНИЙ

Дата

СОДЕРЖАНИЕ

1. Что такое исключение?
2. Виды исключений.
3. Перехват и обработка исключений.
4. Преимущества использования механизма обработки исключений.
5. Резюме.

ЧТО ТАКОЕ ИСКЛЮЧЕНИЕ?

Исключение (exception) – событие, возникающее в ходе выполнения программы, которое разрушает нормальный поток команд программы.

БРОСАНИЕ ИСКЛЮЧЕНИЯ

- В момент возникновения ошибки создаётся *объект исключения*, содержащий информацию об ошибке (включая её тип и место в программе, где она произошла).

- Бросить (throw) исключение - значит создать объект исключения и передать его в систему исполнения.

СТЕК ВЫЗОВОВ МЕТОДОВ

Стек вызовов методов (call stack) – цепочка методов, приводящая к вызову конкретного метода.

ОБРАБОТЧИК ИСКЛЮЧЕНИЙ

- *Обработчик исключения* – блок кода, который может его обработать.

- Выбор подходящего обработчика происходит исходя из типа объекта исключения.

- Поиск начинается с метода, который его бросает, и далее по стеку вызовов в обратном порядке. Если обработчик не будет найден, программа завершится.

ВИДЫ ИСКЛЮЧЕНИЙ

- **Проверяемые**
  - ожидаемые;
  - внутренние;
  - нормальная работа может быть восстановлена.

ВИДЫ ИСКЛЮЧЕНИЙ

- **Проверяемые**
  - ожидаемые;
  - внутренние;
  - нормальная работа может быть восстановлена.

- **Ошибки**
  - неожидаемые;
  - внешние;
  - нормальная работа *не* может быть восстановлена.

ВИДЫ ИСКЛЮЧЕНИЙ

- **Проверяемые**
  - ожидаемые;
  - внутренние;
  - нормальная работа может быть восстановлена.

- **Ошибки**
  - неожидаемые;
  - внешние*;*
  - нормальная работа *не* может быть восстановлена.

- **Непроверяемые**
  - неожидаемые;
  - внутренние;
  - нормальная работа *не* может быть восстановлена.

ИЕРАРХИЯ КЛАССОВ

СТАНДАРТНЫЕ ИСКЛЮЧЕНИЯ

Исключение	Случай использования
IllegalArgumentException	Невалидное значение *не null* параметра
IllegalStateException	Неподходящее состояние объекта для вызова метода
NullPointerException	Значение параметра null, где это запрещено
IndexOutOfBoundsException	Значение индекса за пределами допустимого
ConcurrentModificationException	Обнаружено параллельное изменение объекта, где это запрещено
UnsupportedOperationException	Объект не поддерживает метод

СОБСТВЕННЫЕ КЛАССЫ ИСКЛЮЧЕНИЙ

- Действительно ли вам требуется тип исключения, который не присутствует в Java?
- Поможет ли пользователям тот факт, что они смогут отличать ваше исключение от исключений, бросаемых классами, написанными сторонними разработчиками?
- Будет ли ваше исключение бросаться более чем из одного места?
- Если вы используете чьё-либо другое исключение, будет ли пользователям оно доступно? Должен ли ваш пакет быть независимым и самодостаточным?

ПЕРЕХВАТ И ОБРАБОТКА ИСКЛЮЧЕНИЙ

БЛОК CATCH

- На один блок **try** может быть несколько блоков **catch**.
- Каждый блок **catch** является обработчиком только того типа исключения, который указан в его аргументе.

БЛОК CATCH

В аргументе блока **catch** может быть указано несколько типов, который он обрабатывает. Они разделяются вертикальной чертой (**|**).

Пример:

```
catch (IOException | ItemNotFoundException e)
```

БЛОК CATCH

Система исполнения вызывает первый из блоков, аргумент которого совпадает с типом брошенного исключения.

Пример:

```
} catch (MalformedURLException e)  {
    …
} catch (IOException e)  {
    …
} catch (ItemNotFoundException e)  {
    …
}
```

БЛОК FINALLY

- Всегда исполняется, когда программа выходит из блока **try**, но после исполнения обработчика исключения, если он есть.
- В основном применяется для кода очистки.
- Если JVM заканчивает работу во время выполнения блока **try** или **catch**, то блок **finally** может не исполнится.

TRY С РЕСУРСАМИ

- Try с ресурсами – это выражение **try**, объявляющее один или более ресурсов, которые должны быть закрыты после исполнения блока **try** и обработчика исключения, если он есть.

- Закрываемые ресурсы должны реализовывать интерфейс Autocloseable.

Пример:

```
try (
        ZipFile zipFile = new ZipFile(zipFileName);
        BufferedWriter writer = newBufferedWriter(path, charset)
) {
    …
} catch (IOException e) {
    …
}
```

КАК БРОСИТЬ ИСКЛЮЧЕНИЕ?

Для того, чтобы бросить исключение используйте ключевое слово **throw**.

Пример:

```
throw new FileNotFoundException();
```

КАК БРОСИТЬ ИСКЛЮЧЕНИЕ?

- Используйте ключевое слово **throws** в сигнатуре метода, чтобы задекларировать бросаемые исключения.

  Пример:

```
void someMethod() throws FileNotFoundException {
    …
};
```

- Всегда документируйте бросаемые методом исключения.

ТРЕБОВАНИЕ «CATCH OR SPECIFY»

Если бросаете проверяемое исключение, то:

- либо перехватите его в выражении **try**

```
void someMethod() {
    try {
        …
        throw new SomeException();
    } catch (SomeException e) {
        …
    }
}
```

- либо добавьте его тип в секцию **throws**

```
void someMethod() throws SomeException {
    throw new SomeException();
}
```

ТРЕБОВАНИЕ «CATCH OR SPECIFY»

Метод, который переопределяет метод предка, не может бросать проверяемые исключения, которые не задекларированы в переопределяемом методе или расширяют их.

```
public class SomeException extends Exception {
}
```

```
class A {
  void foo() throws SomeException {
    ...
  }
}

class B extends A {
    void foo() throws Exception {
        …
        throw new Exception();
    }
}
```

```
class A {
  void foo() throws Exception {
    ...
  }
}

class B extends A {
    void foo() throws SomeException {
        …
        throw new SomeException();
    }
}
```

ЦЕПОЧКА ИСКЛЮЧЕНИЙ

Часто бывает удобно ответить на появление одного исключения бросанием другого. Возникает цепочка исключений.

Пример:

```
try {
    …
} catch (IOException e) {
    throw new SampleException("Other  exception", e);
}
```

ЦЕПОЧКА ИСКЛЮЧЕНИЙ

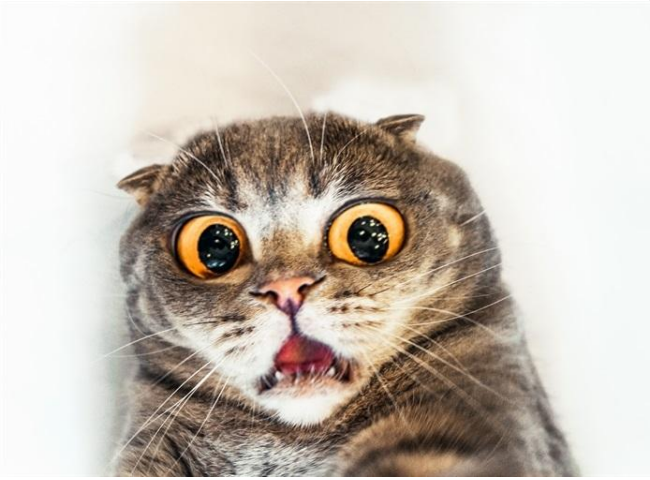
Используйте следующие методы и конструкторы класса Throwable для работы с такими цепочками:

```
Throwable getCause()
Throwable initCause(Throwable)

Throwable(String, Throwable)
Throwable(Throwable)
```

ИСПОЛЬЗУЙТЕ ПРАВИЛЬНО

Исключения для исключительных ситуаций!



```
try {
    int i = 0;
    while (true)  {
        range[i++].climb();
    }
} catch (ArrayIndexOutOfBoundsException e) {
}
```

```
for (Mountain m : range) {
    m.climb();
}
```

ИСПОЛЬЗУЙТЕ ПРАВИЛЬНО

Не забывайте про атомарность! Оставьте объект в том состоянии, в каком он был до вызова метода, бросившего исключение.

ИСПОЛЬЗУЙТЕ ПРАВИЛЬНО

Способы достижения атомарной отказоустойчивости:

1. неизменяемые объекты;
2. проверка параметров на правильность до исполнения основной операции;
3. код восстановления в случае ошибки;
4. исполнить операцию на копии объекта и заменить оригинал объекта на копию в случае успеха.

ПРЕИМУЩЕСТВА

1. Основной код отделяется от кода по обработке ошибок.
2. Передача ошибок выше по стеку вызовов.
3. Группировка и дифференцирование типов ошибок.

РЕЗЮМЕ

- Механизм исключений используется для перехвата и обработки ошибки выполнения программы. Исключения для исключительных ситуаций!
- Все классы исключений являются потомками класса Throwable и описывают тип бросаемого исключения.
- Бывают проверяемые, непроверяемые исключения и ошибки.
- Чтобы бросить исключение используется ключевое слово **throw**.
- Если метод бросает/пробрасывает проверяемое исключение, то он должен задекларировать его в секции **throws**.
- Программа может перехватывать исключения путём использования **try** – **catch** – **finally** блоков.
- Блок **finally** гарантированно будет вызван.

СПАСИБО!