Глава 8
ИСКЛЮЧЕНИЯ И ОШИБКИ

Иерархия и способы обработки

Исключительные ситуации (исключения) возникают во время выполнения программы, когда возникшая проблема не может быть решена в текущем контексте и невозможно продолжение работы программы. Примерами являются особо «популярные»: попытка индексации вне границ массива, вызов метода на нулевой ссылке или деление на нуль. При возникновении исключения создается объект, описывающий это исключение. Затем текущий ход выполнения приложения останавливается, и включается механизм обработки исключений. При этом ссылка на объект-исключение передается обработчику исключений, который пытается решить возникшую проблему и продолжить выполнение программы. Если в классе используется метод, в котором может возникнуть проверяемая исключительная ситуация, но не предусмотрена ее обработка, то ошибка возникает еще на этапе компиляции. При создании такого метода программист должен включить в код обработку исключений, которые могут генерировать этот метод, или передать обработку исключения на более высокий уровень методу, вызвавшему данный метод.

Каждой исключительной ситуации поставлен в соответствие некоторый класс. Если подходящего класса не существует, то он может быть создан разработчиком. Все исключения являются наследниками суперкласса **`Throwable`** и его подклассов **`Error`** и **`Exception`** из пакета **`java.lang`**.

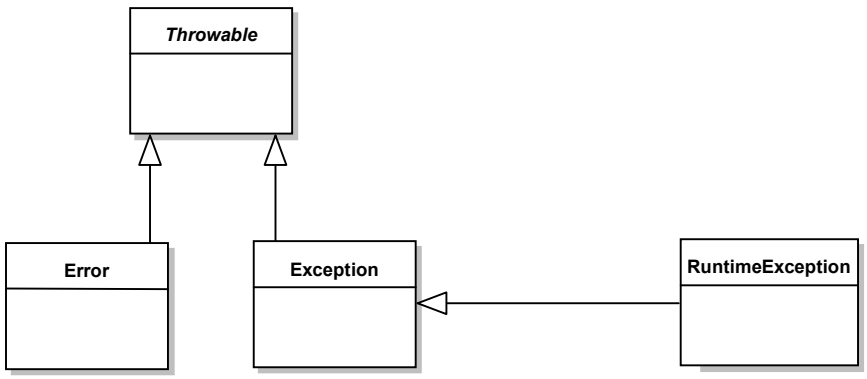

Рис. 8.1. Иерархия основных классов исключений

Исключительные ситуации типа **`Error`** возникают только во время выполнения программы. Такие исключения связаны с серьезными ошибками, к примеру – переполнение стека, и не подлежат исправлению и не могут обрабатываться приложением. Иерархия классов, наследуемых от класса **`Error`**, приведена на рисунке 8.2.

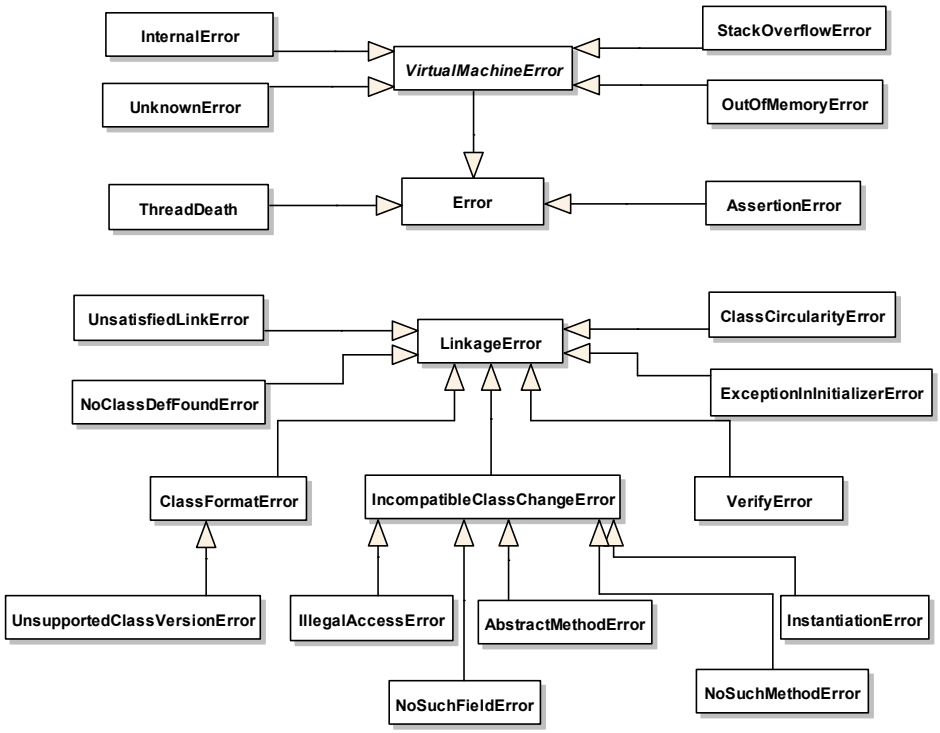

Рис. 8.2. Иерархия классов исключений, наследуемых от класса Error.

На рисунке 8.3 приведена иерархия классов исключений, наследуемых от класса **Exception**.

Проверяемые исключения должны быть обработаны в методе, который может их генерировать, или включены в **throws**-список метода для дальнейшей обработки в вызывающих методах. Возможность возникновения проверяемого исключения может быть отслежена на этапе компиляции кода.

Во время выполнения могут генерироваться также исключения, которые могут быть обработаны без ущерба для выполнения программы. Иерархия этих исключений приведена на рисунке 8.4. В отличие от проверяемых исключений, класс **RuntimeException** и порожденные от него классы относятся к непроверяемым исключениям. Компилятор не проверяет, генерирует ли и обрабатывает ли метод эти исключения. Исключения типа **RuntimeException** автоматически генерируются при возникновении ошибок во время выполнения приложения. Таким образом, нет необходимости в проверке генерации исключения вида:

```
if(a==null) throw new NullPionterException();
```

объект класса **NullPionterException** при возникновении ошибки будет сгенерирован автоматически. Кроме этого, в любом случае нет необходимости в обработке этого исключения непосредственно в методе или в передаче на обработку вызывающему методу с помощью оператора **throw**. В конце концов исключение будет передано в метод **main()**, где обрабатывается вызовом метода **printStackTrace()**, выдающего данные трассировки.

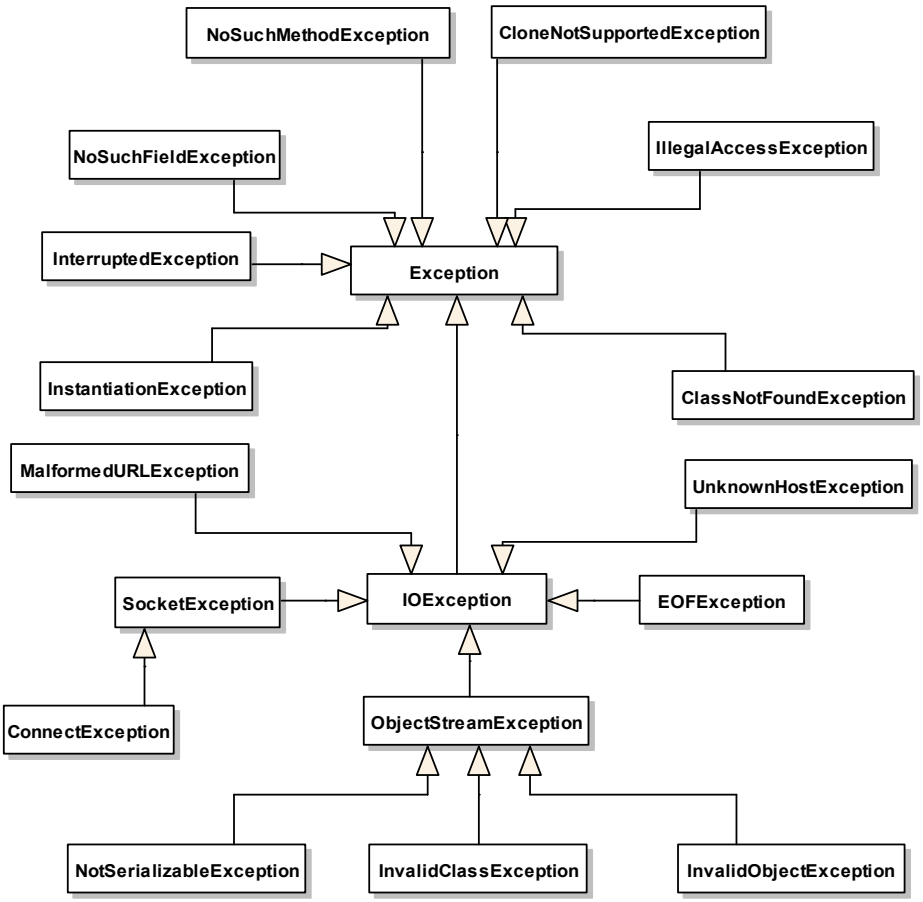

Рис. 8.3. Иерархия классов проверяемых (checked) исключительных ситуаций

Обычно используется один из трех способов обработки исключений:

- перехват и обработка исключения в блоке **`try-catch`** метода;
- объявление исключения в секции **`throws`** метода и передача вызывающему методу (для проверяемых исключений);
- использование собственных исключений.

Первый подход можно рассмотреть на следующем примере. При клонировании объекта в определенных ситуациях может возникать исключительная ситуация типа **`CloneNotSupportedException`**. Например:

```
public void changeObject(Student ob) {
            try {
                  Object temp = ob.clone();
                  //реализация
            } catch (CloneNotSupportedException e) {
                  System.err.print(e);
            }
      }
```

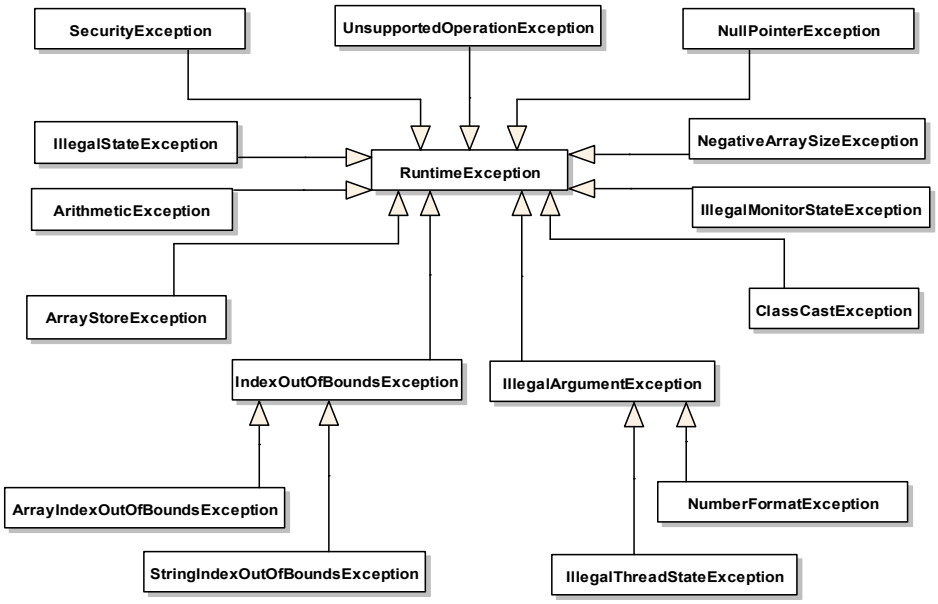

Рис. 8.4. Иерархия непроверяемых (unchecked) исключений

При клонировании может возникнуть исключительная ситуация в случае, если переданный объект не поддерживает клонирование (не включен интерфейс **`Cloneable`**). В этом случае генерируется соответствующий объект, и управление передается блоку **`catch`**, в котором обрабатывается данный тип исключения, иначе блок **`catch`** пропускается. Блок **`try`** похож на обычный логический блок. Блок **`catch(){}`** похож на метод, принимающий в качестве единственного параметра ссылку на объект-исключение и обрабатывающий этот объект.

Второй подход демонстрируется на этом же примере. Метод может генерировать исключения, которые сам не обрабатывает, а передает для обработки другим методам, вызывающим данный метод. В этом случае метод должен объявить о таком поведении с помощью ключевого слова **`throws`**, чтобы вызывающие методы могли защитить себя от этих исключений. В вызывающих методах должна быть предусмотрена обработка этих исключений. Форма объявления такого метода:

```
Тип имяМетода(список_аргументов)
        throws список_исключений { }
```

При этом сам таким образом объявляемый метод может содержать блоки **`try-catch`**, а может и не содержать их. Например, метод **`changeObject()`** можно объявить:

```
public void changeObject(Student ob)
            throws CloneNotSupportedException {
            Object temp = ob.clone();
                  //реализация
            }
```

Ключевое слово **`throws`** позволяет разобраться с исключениями методов «чужих» классов, код которых отсутствует. Обрабатывать исключение при этом будет метод, вызывающий **`changeObject()`**:

```
public void load(Student stud) {
     try {
          changeObject(stud);
     } catch (CloneNotSupportedException e) {
          String error = e.toString();
          System.err.println(error);
     }
}
```

Создание собственных исключений будет рассмотрено позже в этой главе.

Если в блоке **try** может быть сгенерировано в разных участках кода несколько типов исключений, то необходимо наличие нескольких блоков **catch**, если только блок **catch** не обрабатывает все типы исключений.

```
/* пример # 1 : обработка двух типов исключений: TwoException.java */
package chapt08;

public class TwoException {
     public static void main(String[] args) {
          try {
               int a = (int)(Math.random() * 2);
               System.out.println("a = " + a);
               int c[] = { 1/a };
               c[a] = 71;
          } catch(ArithmeticException e) {
               System.err.println("деление на 0" + e);
          } catch(ArrayIndexOutOfBoundsException e) {
               System.err.println(
                       "превышение границ массива: " + e);
          }//последний catch
               System.out.println("после блока try-catch");
     }
}
```

Исключение "деление на 0" возникнет при инициализации элемента массива **a=0**. В противном случае (при **a=1**) генерируется исключение "превышение границ массива" при попытке присвоить значение второму элементу массива **c[]**, который содержит только один элемент. Однако пример, приведенный выше, носит чисто демонстративный характер и не является образцом хорошего кода, так как в этой ситуации можно было обойтись простой проверкой аргументов на допустимые значения перед выполнением операций. К тому же генерация и обработка исключения – операция значительно более ресурсоемкая, чем вызов оператора **if** для проверки аргумента. Исключения должны применяться только для обработки исключительных ситуаций, и если существует возможность обойтись без них, то следует так и поступить.

Подклассы исключений в блоках **catch** должны следовать перед любым из их суперклассов, иначе суперкласс будет перехватывать эти исключения. Например:

```
try {/*код, который может вызвать исключение*/
} catch(RuntimeException e) { /* суперкласс RuntimeException
                                 перехватит объекты всех своих подклассов */
```

```
} catch(ArithmeticException e) {} /* не может быть вызван,
                                     поэтому возникает ошибка компиляции */
```

Операторы **try** можно вкладывать друг в друга. Если у оператора **try** низкого уровня нет раздела **catch**, соответствующего возникшему исключению, поиск будет развернут на одну ступень выше, и будут проверены разделы **catch** внешнего оператора **try**.

```
/* пример # 2 : вложенные блоки try-catch: MultiTryCatch.java */
package chapt08;

public class MultiTryCatch {
     public static void main(String[] args) {
          try { // внешний блок
               int a = (int) (Math.random() * 2) - 1;
               System.out.println("a = " + a);
               try { // внутренний блок
                    int b = 1/a;
                    StringBuffer sb = new StringBuffer(a);
               } catch (NegativeArraySizeException e) {
                    System.err.println(
                         "недопустимый размер буфера: " + e);
               }
          } catch (ArithmeticException e) {
               System.err.println("деление на 0" + e);
          }
     }
}
```

В результате запуска приложения при **a=0** будет сгенерировано исключение **ArithmeticException**, а подходящий для его обработки блок **try-catch** является внешним по отношению к месту генерации исключения. Этот блок и будет задействован для обработки возникшей исключительной ситуации.

Третий подход к обработке исключений будет рассмотрен ниже на примере создания пользовательских исключений.

Оператор throw

В программировании часто возникают ситуации, когда программисту необходимо самому инициировать генерацию исключения для указания, например, на заведомо ошибочный результат выполнения операции, на некорректные значения параметра метода и др. Исключительную ситуацию можно создать с помощью оператора **throw**, если объект-исключение уже существует, или инициализировать его прямо после этого оператора. Оператор **throw** используется для генерации исключения. Для этого может быть использован объект класса **Throwable** или объект его подкласса, а также ссылки на них. Общая форма записи инструкции **throw**, генерирующей исключение:

```
throw объектThrowable;
```