Обработка исключительных ситуаций

Синтаксис

```
try{
    операторы0;
} catch (ТипИсключения1 переменная1){
    операторы1;
} catch (ТипИсключения2 переменная2){
    операторы2;
} catch (ТипИсключенияN переменнаяN){
    операторыN;
} finally{
    операторы;
}
```

Пример

```
void myETest(String s,double y){
    double x, z;
    try{
        x=Double.parseDouble(s);
        z=Math.sqrt(x/y);
    } catch(ArithmeticException e){
    System.out.println("Деление на ноль или корень из
    отрицательного числа ");
    } catch(NumberFormatException e){
    System.out.println("Ошибка преобразования!");
    }
};
```

Иерархия исключительных ситуаций

- Throwable

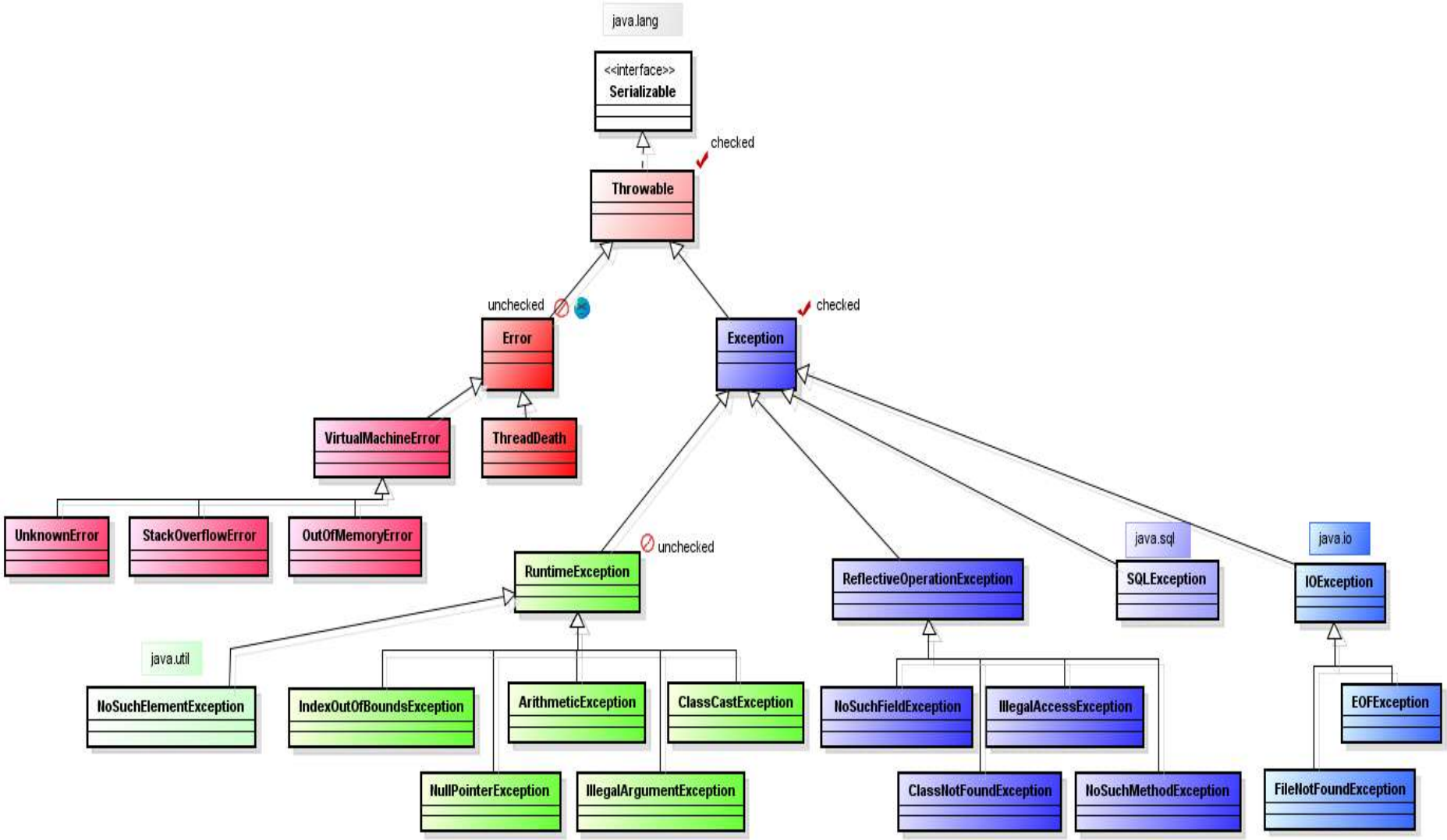
  Error

  Потомки – ***непроверяемые*** исключительные ситуации

  Exception

  Потомки – ***проверяемые*** исключительные ситуации

  RuntimeException и его потомки – ***непроверяемые*** исключительные ситуации


java.lang
<<interface>>
Serializable
checked
Throwable
unchecked
Error
checked
Exception
VirtualMachineError
ThreadDeath
UnknownError
StackOverflowError
OutOfMemoryError
unchecked
RuntimeException
java.sql
java.io
ReflectiveOperationException
SQLException
IOException
java.util
NoSuchElementException
IndexOutOfBoundsException
ArithmeticException
ClassCastException
NullPointerException
IllegalArgumentException
NoSuchFieldException
IllegalAccessException
ClassNotFoundException
NoSuchMethodException
EOFException
FileNotFoundException

Иерархия исключительных ситуаций

- Непроверяемые исключения генерируются и обрабатываются системой автоматически – как правило, приводя к завершению приложения. При этом их типы нигде не указываются, и слово throws в заголовке метода указывать не надо.

- Если в теле реализуемого метода используется вызов метода, который может возбуждать исключительную ситуацию, и это исключение не перехватывается, в заголовке реализуемого метода требуется указывать тип возбуждаемого исключения в списке throws .

Собственные исключения

```
class WrongPasswordException extends Exception {
    public WrongPasswordException(){ // конструктор
        System.out.println(”Wrong password!”);
    }
}
class MyErrException extends Exception {
private String mes;
    public WrongPasswordException(String s){ // конструктор
        mes=s;
    }
  public String toString() {
    return mes;
 }
}
```

Возбуждение исключения

```
throw new WrongPasswordException();
throw new MyErrException(“error”);
throw new MyErrException(“not equal”);
```

Объявление метода, возбуждающего исключение.

Тип имя(*список параметров*) **throws**
ТипИсключения1, ТипИсключения2,…,
ТипИсключенияN{

Тело функции, содержащее оператор throw или вызов метода, бросающего исключение, но не перехватывающее его

}

Пример

```
class CheckPasswordDemo{
    private String password="";
    public String getPassword(){
        return password;
    }
    public void setPassword()(){
        //реализация метода
    }
    public void checkPassword(String pass)
          throws  WrongPasswordException {
        if(!pass.equals(password))
              throw new WrongPasswordException();
    }
}
```

- При обработке исключения в блоке catch можно получить отладочную информацию, полезную для разработчика. Для этого можно воспользоваться методами getMessage() и printStackTrace() класса Throwable
- Метод printStackTrace() распечатывает *трассировку стека. Трассировка стека – это список* вызовов методов для данной точки программы.
- В версиях Java, начиная с Java SE 1.4, для получения описания трассировки стека разработчикам доступен метод getStackTrace(). Он возвращает массив объектов StackTraceElement, которые можно анализировать в программе

Java 7

```
try {
    ...
  } catch( IOException | SQLException ex ) {
    … //проверка и реакция
  }
```

Завершающие действия

```
try {
    ...
   } catch(  . . .) {
     // Что здесь может быть???


} finally {
   //выполняется независимо  было ли исключение или
   // нет
   // здесь нужно закрыть файлы и сетевые ресурсы
}
```

Java 7

- TWR (try-with-resources)

```
try (BufferedReader br =
        new BufferedReader(new FileReader(path)))
{
        return br.readLine();
}
// блоки catch и finally могут быть опущены
```

Java 7

- TWR (try-with-resources)
- в заголовке блока try можно указать несколько ресурсов (через «;»)
- если опущен блок catch – исключение не обрабатывается в методе, а выбрасывается
- если используется блок finally в нем не нужно закрывать ресурсы, указанные в заголовке блока try