

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 отриц-го числа ");
    } catch(NumberFormatException e){
        System.out.println("Ошибка преобразования!");
    }
};
```

Иерархия исключительных ситуаций

- Throwable

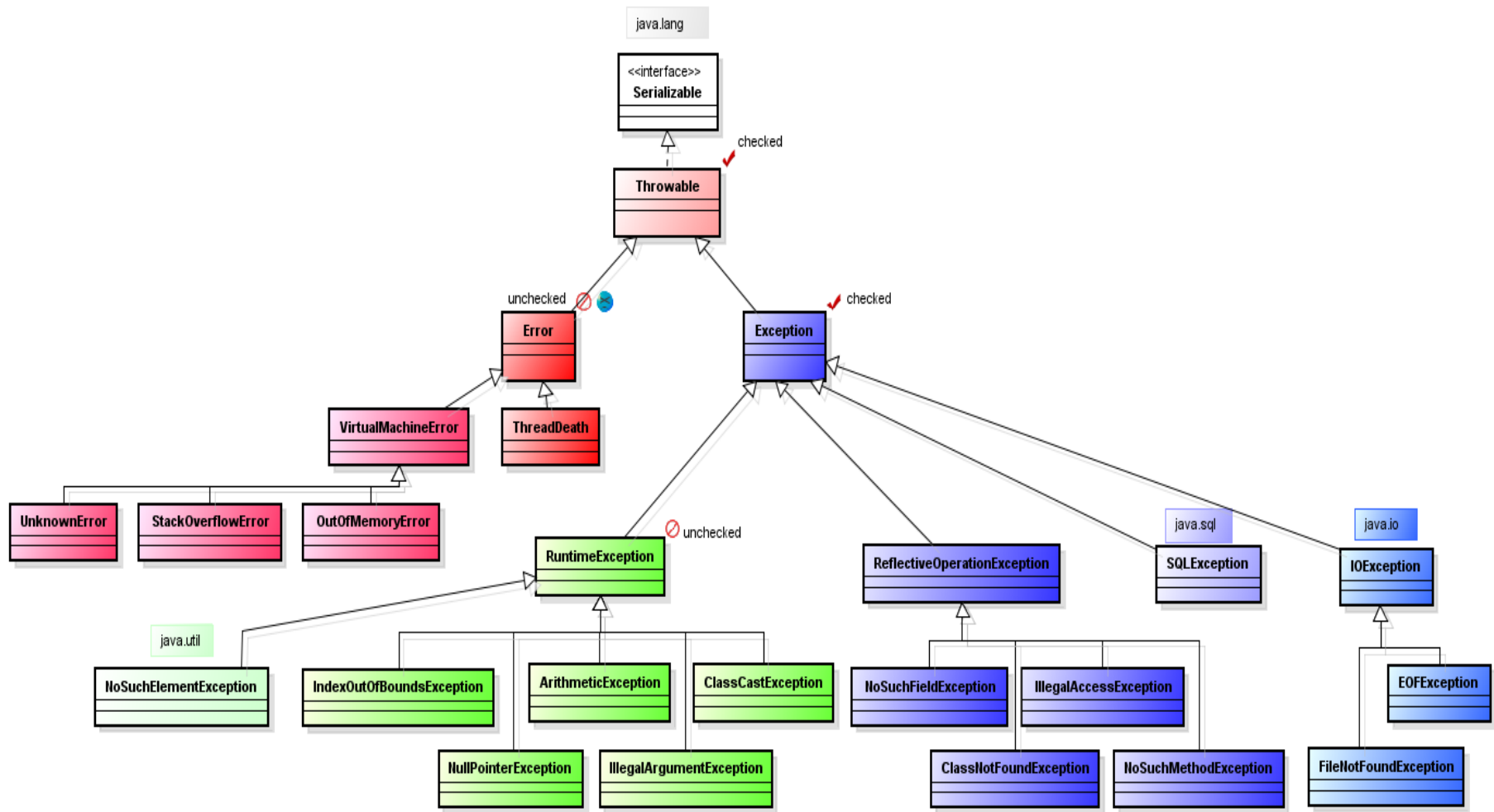
Error

Потомки – **непроверяемые** исключительные ситуации

Exception

Потомки – **проверяемые** исключительные ситуации

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



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 проверяемую исключительную ситуацию, и это исключение не перехватывается, в заголовке реализуемого метода требуется указывать тип возбуждаемого исключения в списке `throws` .

Иерархия исключительных ситуаций

Throwable

- Получает снимок стека выполнения на момент ошибки
- Может также получать дополнительную информацию о причине в виде строки

Error

- Серьезные проблемы, которые не имеет смысла обрабатывать приложению

Exception

- Состояния, которые возможно обрабатывать в приложении
 - RuntimeException не требуют явной обработки
 - Остальные требуют явной обработки или сообщения о том, что обработка невозможна

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;  
    }  
}
```


Выбрасывание исключения

```
throw new WrongPasswordException();  
throw new MyErrException("error");  
throw new MyErrException("not equal");  
throw new Exception("Error in index");
```

Объявление метода, выбрасывающего ИСКЛЮЧЕНИЕ

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

{

*Тело метода, содержащее оператор throw или вызов метода,
бросающего исключение, но не обрабатывающее 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

Объединение нескольких исключительных ситуаций, требующих одинаковую обработку

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

JUnit 4

- Аннотации
 - @Test
 - @Test (expected = Класс_исключения.class)
 - @Test(timeout=миллисекунд)
 - @Ignore
 - @Ignore(“комментарий”)

JUnit 4

- Методы класса Assert
 - assertEquals()
 - assertTrue() / assertFalse()
 - assertEquals() / assertEquals()
 - assertNull() / assertNotNull()
 - fail()

JUnit 4

- Фикстуры
 - @BeforeClass
 - @AfterClass
 - @Before
 - @After