Практическая работа № 19. Создание пользовательских исключений

Цель данной практической работы – научиться создавать собственные исключения.

Теоретические сведения

Язык Java предоставляет исчерпывающий набор классов исключений, но иногда при разработке программ вам потребуется создавать новые – свои собственные исключения, которые являются специфическими для потребностей именно вашего приложения. В этой практической работе вы научитесь создавать свои собственные пользовательские классы исключений. Как вы уже знаете, в Java есть два вида исключений-проверяемы и непроверяемые. Для начала рассмотрим создание пользовательских проверяемых исключений.

Создание проверяемых пользовательских исключений

Проверяемые исключения — это исключения, которые необходимо обрабатывать явно. Рассмотрим пример кода:

Листинг 19.1 Пример обработки исключения

```
try (Scanner file = new Scanner(new File(fileName))){
   if (file.hasNextLine()) return file.nextLine();
} catch(FileNotFoundException e) {
   // Logging, etc
}
```

Приведенный на листинге 19.1 код является классическим способом обработки проверяемых исключений на Java. Хотя код выдает исключение FileNotFoundException, но в целом неясно, какова точная причина ошибки – не такого файла нет или же имя файла является недопустимым.

Чтобы создать собственное пользовательское исключение, мы будем наследоваться от класса java.lang.Exception. Давайте рассмотрим пример как это реализуется на практике и создадим собственный класс для проверяемого исключения с именем BadFileNameException:

Листинг 19.2 Пример класса исключения

```
public class BadFileNameException extends Exception {
```

```
        public BadFileNameException(String
errorMessage){
            super(errorMessage);
        }
    }
```

Обратите внимание, что мы также должны написать конструктор в нашем классе, который принимает параметр типа String в качестве сообщения об ошибке, в котором вызывается конструктор родительского класса. Фактически это все, что нам нужно сделать, чтобы определить свое собственное пользовательское исключение.

Далее, давайте посмотрим, как мы можем использовать пользовательское исключение в программе

Листинг 19.3 Пример генерации исключения из за неправильного имени файла

```
    try (Scanner file = new Scanner(new
File(fileName))){
        if (file.hasNextLine())
            return file.nextLine();
    } catch (FileNotFoundException e) {
        if (!isCorrectFileName(fileName)) {
            throw new BadFileNameException("Bad
filename : " + fileName );
        }
        //...
    }
```

Мы создали и использовали свое собственное пользовательское исключение, теперь в случае ошибки, можно понять, что произошло, и какое именно исключение сработало. Как вы думаете, этого достаточно? Если ваш ответ да, то мы не узнаем основную причину, по которой сработало исключения. Как исправить программу. Для этого мы также можем добавить параметр java.lang.Throwable в конструктор. Таким образом, мы можем передать родительское исключение во время вызова метода:

Листинг 19.4 Пример использования исключения

```
    public BadFileNameException(String errorMessage,
Throwable err) {
        super(errorMessage, err);
    }
```

Теперь мы связали BadFileNameException с основной причиной возникновения данного исключения, например:

Листинг 19.5 Пример обработки исключительной ситуации некорректного имени файла

```
    try (Scanner file = new Scanner(new
File(fileName))){
        if (file.hasNextLine()) {
            return file.nextLine();
        }
    } catch (FileNotFoundException err) {
        if (!isCorrectFileName(fileName)) {
            throw new BadFileNameException(
              "Bad filename: " + fileName, err);
        }
        // ...
    }
```

Мы рассмотрели, как мы можем использовать пользовательские исключения в программах, учитывая их связь с причинами по которым они могут возникать.

Создание непроверяемых пользовательских исключений

В том же примере, который мы рассматривали выше предположим, что нам нужно такое пользовательское исключение, в котором обрабатывается ошибка, если файла не содержит расширения.

В этом случае нам как раз понадобится создать пользовательское непроверяемое исключение, похожее на предыдущее, потому что данная ошибка будет обнаружена только во время выполнения участка кода. Чтобы создать собственное непроверяемое исключение, нам нужно наследоваться от класса java.lang.RuntimeException:

Листинг 19.6 Пример создания пользовательского класса исключения

```
public class BadFileExtensionException
  extends RuntimeException {
```

```
        public BadFileExtensionException(String
errorMessage, Throwable err) {
            super(errorMessage, err);
        }
    }
```

Теперь, мы можем использовать это нестандартное исключение в рассматриваемом нами выше примере:

Листинг 19.7 Пример использования нестандартного исключения

```
    try (Scanner file = new Scanner(new
File(fileName))){
        if (file.hasNextLine()) {
            return file.nextLine();
        } else {
            throw new IllegalArgumentException("Non
readable file");
        }
    } catch (FileNotFoundException err) {
        if (!isCorrectFileName(fileName)) {
            throw new BadFileNameException(
              "Bad filename: " + fileName , err);
        }

        //...
    } catch(IllegalArgumentException err) {
        if(!containsExtension(fileName)) {
            throw new BadFileExtensionException(
              "Filename does not contain extension: "
+ fileName, err);
        }

        //...
    }
```

Заключение

В приведенных выше примерах мы рассмотрели основные особенности обработки исключений.

Задания на практическую работу № 19

1. Клиент совершает покупку онлайн. При оформлении заказа у пользователя запрашивается фио и номер ИНН. В программе проверяется, действителен ли номер ИНН для такого клиента. Исключение будет выдано в том случае, если введен недействительный ИНН.

2. Предлагается модернизировать задачу из предыдущей практической работы (см. методические указания по выполнению практических работ №1-8) – задача сортировки студентов по среднему баллу. Необходимо разработать пользовательский интерфейс для задачи поиска и сортировки (использовать массив интерфейсных ссылок- пример в лекции 5). Дополнить ее поиском студента по фио – в случае отсутствия такого студента необходимо выдавать собственное исключение. Схема классов прогоаммы приведена на Рис.19.1.

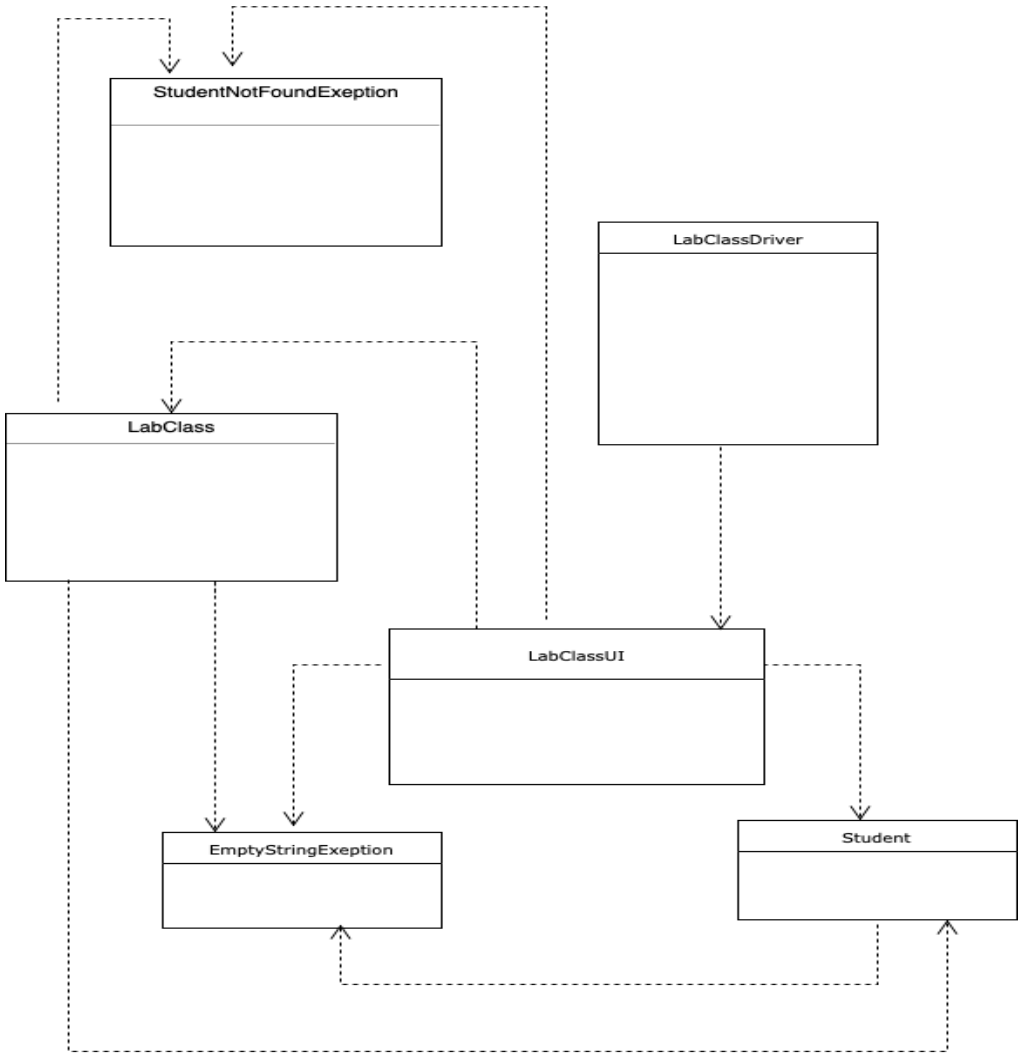

Рисунок 19.1. UML диаграмма проекта LabClass с обработкой исключений

Ссылки на источники

1. http://www.embeddedsystemonline.com/programming-languages/java/10-java-exceptions
2. http://java.sun.com/docs/books/tutorial/essential/exceptions/index.html
3. https://habrahabr.ru/company/golovachcourses/blog/223821/
4. http://kostin.ws/java/java-exceptions.html