

April 25th, 2022

Sheet 04 – Unit Testing, Debugging, Exception Handling

Exercise 1 (Propper error handling)

[2 Points]

- Get familiar with the code provided in `JavaErrorhandling.java`^{OLAT}. Replace the clumsy error handling with the object oriented error handling approach discussed in the lecture.
- Get familiar with the code provided in `MyLinkedList.java`^{OLAT} and `Node.java`^{OLAT}. The provided code represents a simple linked list that only supports inserting at the end, querying the list size and getting the element at the given index.

Read through the don'ts with regard to exception handling on slide 45 and 46 of PM-13-Exceptions.pdf^{OLAT}. Modify the code of `MyLinkedList` until it is compliant with what you just read.

Try looking for appropriate exceptions/runtime exceptions in the java API, if you don't find any you can create your own exceptions.

Submit



```
at/ac/uibk/pm/gXX/zidUsername/s04/e01/JavaErrorhandling.java
at/ac/uibk/pm/gXX/zidUsername/s04/e01/MyLinkedList.java
at/ac/uibk/pm/gXX/zidUsername/s04/e01/Node.java
...
```

Exercise 2 (Exceptions)

[1 Point]

The goal of this exercise is to look at the given code, get familiar with it, and handle Exceptions. Take a look at the provided code in `Course.java`^{OLAT}. This course contains an `addStudent` method that returns `true` if a student could be added and `false` otherwise. Notice that the code returns `false` in two cases. One case is that the course has no available places left and the other is that a student is already in the course. Modify the given code to make it possible to distinguish between these two cases using exception handling in the given main method. Therefore, you need to implement a `CourseFullException` and a `StudentAlreadyEnrolledException`^{OLAT}. Modify the `Course.java` and the `CourseApplication.java`^{OLAT} such that it additionally states the reason why a student could not be added in case this happens.

Submit

```
at/ac/uibk/pm/gXX/zidUsername/s04/e02/Course.java
at/ac/uibk/pm/gXX/zidUsername/s04/e02/CourseApplication.java
at/ac/uibk/pm/gXX/zidUsername/s04/e02/CourseFullException.java
at/ac/uibk/pm/gXX/zidUsername/s04/e02/StudentAlreadyEnrolledException.java
```

Exercise 3 (Unit Testing)**[4 Points]**

Testing is a very important part of programming. If you neglect it, you could regret it. On this exercise you will put in practice your knowledge about Unit Testing. To do so, implement a simple password-checker, which determines the strength of a password. Consider the following password-rules:

- It must be at least 16 characters long.
- It must contain at least two numbers.
- It must contain at least three lowercase characters.
- It must contain at least two uppercase characters.
- It must contain at least two of the following characters: `?, !, %, &, =, [,], +, -`. If a illegal character is entered an Exception is thrown.
- It cannot contain the first name nor the last name nor the day, month or year of birth of the user.

Your job is to provide an implementation according to the following method signature:

```
public PasswordStrength checkPassword(User user, String password) throws...
```

Now consider the following requirements:

- If parameters are incorrect (e.g.: null-values, not every attribute is set at the user-parameter, empty password, unexpected illegal special character, etc...), an adequate Exception should be thrown.
- A password is `TOO_WEAK` if it satisfies less than 2 rules, `WEAK` if it satisfies at most two rules. It is `MEDIUM` if it satisfies more than two, but less than five rules. A password is `STRONG` if it satisfies five rules or more.

- a) **2.0 Points** Implement the password-checker according to the described requirements. Also define necessary methods and choose appropriate visibility of your attributes in order to follow data-encapsulation principles. Additionally, you have to submit all the necessary files (e.g., a class that represents a user, etc.) to demonstrate their functionality.

Submit


```
at/ac/uibk/pm/gXX/zidUsername/s04/e03/PasswordChecker.java
at/ac/uibk/pm/gXX/zidUsername/s04/e03/PasswordStrength.java
...
```

- b) 2.0 Points Now check your implementation by providing adequate JUnit-tests. Try to cover all aspects of your implementation with your tests. The number of tests is up to you, but they should cover at least the following requirements:

- Test with invalid parameters (and check if the Exceptions are thrown as expected)
- Test each password-rule at least once (structure your code accordingly)
- There must be a test for each 'password-strength' (TOO_WEAK, WEAK, MEDIUM, and STRONG).
- Also test the boundaries between the levels of strength

Submit



 at/ac/uibk/pm/gXX/zidUsername/s04/e03/PasswordCheckerTest.java

 ...

Exercise 4 (Exceptions and Testing)

[3 Points]


The goal of this exercise is to apply your gained knowledge about handling exceptions and managing unit testing. In this exercise you have to get familiar with the code and provide tests and implement exceptions when they are asked.


- a) Given the class `Item.java` ^{OLAT} create JUnit-tests for the following cases:
- test whether the constructor throws a `IllegalArgumentException` if the name is null or if the price of the item is smaller or equal to 0
 - test whether the name is correctly returned
 - test whether the price is correctly returned


Change the class `Item.java` ^{OLAT} implementation such that these test are successful.


- b) Create a class `ItemManager`. This class manages a list of items with a predefined maximum size. The item manager should provide:
- a `size` method that returns the amount of items stored in the item manager.
 - an `addItem` method to add items, which throws an exception if max size is exceeded.
 - a `removeItem` method to remove an item, which throws an exception if the item is not contained.
 - a `totalPrice` method to get the total value of the stored items.
- c) Create a class `ItemManagerTest` using JUnit and add tests for the following cases:
- test whether the constructor throws an `IllegalArgumentException` if the size is smaller than zero.
 - test whether items are correctly added.
 - test whether no more items can be added if the item manager reaches the maximum size and that an exception is thrown in such a case.
 - test whether items are correctly removed.
 - test whether the total price is correctly calculated.


Submit

 at/ac/uibk/pm/gXX/zidUsername/s04/e04/Item.java

 at/ac/uibk/pm/gXX/zidUsername/s04/e04/ItemTest.java

 at/ac/uibk/pm/gXX/zidUsername/s04/e04/ItemManager.java

 at/ac/uibk/pm/gXX/zidUsername/s04/e04/ItemManagerTest.java

 ...

Important: Submit your solution to OLAT and mark your solved exercises with the provided checkboxes. The deadline ends at 6:00 pm (18:00) on the day before the discussion.