

Java CA Assignment Report

Dervla Scully

18329511

For my solution to the assignment I have a JUnit test class name 'TestCheckedUser' which tests a given solution to the exam question and shows all tests which failed/ passed. I also wanted to be able to calculate and display the grade received for each solution to the exam question so I have done so in the class 'grade'. I have made three projects, each of which contain the classes CheckedUser (CheckedUser is slightly different for each so that one is a B or above solution, one is a C solution and one is a D solution), TestCheckedUser, grade and main (which are all the exact same for each of the three projects). In this report I will explain in detail the components of each of these classes.

(Note: report is slightly longer than the required 5 pages as I included a large table and had to explain both TestCheckedUser.java and grade.java)

TestCheckedUser.java:

When TestCheckedUser.java is ran it shows all tests which passed and failed. Each test is explained below.

checkGetName()

This test checks that the getName() method in CheckedUser returns the name for a given instance of a CheckedUser correctly. Inside a try block I made an instance of CheckedUser called testUser using the constructor for just name and assigned it the name "John Scully". Then using assertEquals I tested whether the String returned from getName() for the testUser was equal to the string "John Scully". I then had a catch block which will catch any exception thrown. If the strings are not equal, the test will fail. If the strings are equal, the test will pass.

checkGetAge()

In the exact same way as above I tested the getAge() method for CheckedUser. I assigned the value for the age for the instance of CheckedUser (testUser) as 22, and then tested whether this was equal to the int returned from testUser.getAge(). I ensured that the age entered was a valid age to ensure that no other exceptions would be thrown (e.g. if age was below 18 or above 30 an exception would be thrown)

checkGetEmail()

Exact same as above tests but assigning the value for the age for the instance of CheckedUser (testUser) as "john.scully@ucd.ie" and testing whether this was equal to testUser.getEmail(). Again like above I ensured that no other exceptions would be thrown by entering name, age and email according to the criteria in part (c) of the question.

checkGetEmailLowerCase()

The email should be all lower case, as part (c) states "email addresses should be stored in lower case even if provided otherwise", so in this test I have given "John.Scully@ucd.ie" as the email for the testUser and am testing in the same way as the test above whether testUser.getEmail() is equal to "john.scully@ucd.ie" (ie the email converted to lower case).

checkSetName()

In this method I am testing the setName() method for CheckedUser. Inside of a try block I have created an instance of CheckedUser called testUser and am giving it the name "John Scully". I am then setting the name of testUser to "Jack". I then test using assertEquals whether testUser.getName() is equal to "Jack". If the setName() method is correct these should be equal as the instance variable name for testUser should have been changed to now hold the string "Jack".

checkSetAge()

In the exact same way as above, to test the method `setAge()` I have made an instance of `CheckedUser` called `testUser`, and in this case using the constructor for just name. I then set the age for test user to 24 using `testUser.setAge()`. Then again using `assertEquals` I have tested whether `testUser.getAge()` is equal to 24.

checkSetEmail()

In the same way as above I am testing the `setEmail()` method by making of `CheckedUser` called `testUser`, assigning it the email "john.scully@ucdconnect.ie" using `setEmail()`, and testing whether this is equal to `testUser.getEmail()` using `assertEquals`.

checkNameConstructor()

In this test I am testing the constructor for just name by creating an instance of `CheckedUser` with just name in a try block and catching any exceptions in a catch block.

checkNameAgeConstructor()

In the same way as above I am testing the constructor for name and age by creating an instance of `CheckedUser` with name and age in a try block and catching any exceptions in a catch block.

checkNameAgeEmailConstructor()

In the same way as above I am testing the constructor for name, age and email by creating an instance of `CheckedUser` with name, age and email in a try block and catching any exceptions in a catch block.

testCheckNameNull()

I am testing to ensure an illegal argument exception is thrown if an instance of `CheckedUser` is created with name equal to "" / empty string.

I first made a variable called `thrown` of type boolean and set it to false. Then inside a try block I made an instance of `CheckedUser` called `testUser` and set its value for name to "". If the `CheckedUser` solution is correct this should throw an exception. I have a catch block which should catch an `IllegalArgumentException`. If it does then the variable `thrown` is set to true. Then after the catch block I have an `assertTrue` that tests whether the variable `thrown` holds the value true. If the `CheckedUser` solution was correct an `IllegalArgumentException` would have been thrown and the variable `thrown` would have been set to true. Therefore, the `assertTrue(thrown)` would pass. If the solution was incorrect in that an `IllegalArgumentException` was not thrown, `thrown` would still hold the value false and the `assertTrue(thrown)` would fail.

testCheckNamePrefix1() - testCheckNamePrefix5()

In these 5 tests I am testing to ensure that an illegal argument exception is thrown if an instance of `CheckedUser` is constructed with a prefix (Mr, Mrs etc) before the name. In each I made an instance of `CheckedUser`, giving each a different prefix before the name (1: Mr, 2: Mrs, 3: Miss, 4: Ms, 5: Dr). If the solution for `CheckedUser` is correct then each of these should throw an `IllegalArgumentException`. I tested this the exact same way as explained above using the boolean variable `thrown`, setting it to false, setting it to true in the catch block and then doing `assertTrue(thrown)`.

testCheckNameLength()

In this test I am testing to ensure that an illegal argument exception is thrown if an instance of `CheckedUser` is constructed with a name with more than two words. I constructed an instance of `CheckedUser` with 4 words in the name. I tested if an illegal argument exception was thrown in the same way as a previous two tests above.

testCheckAgeRange1() & testCheckAgeRange2()

In these tests I am testing to ensure that an illegal argument exception is thrown if an instance of `CheckedUser` is constructed with age less than 18 or greater than 30. In each of these tests I constructed an instance of `CheckedUser`, one with age 17 and one with age 31. I tested if an illegal argument exception was thrown in the same way as a previous three tests above.

testCheckEmailMatchesName1() & testCheckEmailMatchesName2()

In these tests I am testing to ensure that an illegal argument exception is thrown if an instance of CheckedUser is constructed with an email that doesn't match the name, as email should be of the form firstName.lastName@<somewhere>.<something> (if name is two words) or firstName.lastName@<somewhere>.<something> (if name is one word). In each test I am constructing an instance of CheckedUser, and in the first test assigning it a name which is two words, and in the second a name that is one word. I assigned both an email which does not match the same, therefore an exception should be thrown. I have tested this in the exact same way as above 4 tests.

testCheckEmailDomainNull1() & testCheckEmailDomainNull1()

In these tests I am testing to ensure that an illegal argument exception is thrown if an instance of CheckedUser is constructed with an email domain that is blank, ie "" or " ". The email domain is the <somewhere> in <firstName>.<lastName>@<somewhere>.<something>. To test this I have made an instance of CheckedUser in each, giving an email with a domain of "" to one and " " to the other (ie john.scully@.ie and john.scully@.ie) I have tested this in the exact same way as the 5 tests above.

testCheckEmailDomainSpecialCharacter()

As well as the criteria for name, age and email to be valid which was given in part (c), I tried to think of other cases / edge cases in which they would not be valid. One of which would be if special characters (*&%^ etc) were present in the email domain. I made an instance of CheckedUser in each, giving an email with a domain containing special characters. If the solution is correct in that it takes into account this edge case then it should throw an exception here. I tested this in the exact same way as the 6 tests above.

testCheckEmailDomainExtensionNull1() & testCheckEmailDomainExtensionNull2()

Another edge case would be if the domain extension (ie the <something> in <firstName>.<lastName>@<somewhere>.<something>, e.g. .com, .ie) was empty (ie "" or " "). To test this I have made an instance of CheckedUser in each, giving an email with a domain extension of "" to one and " " to the other. If the solution being tested is correct then these should throw IllegalArgumentExceptions. I have tested these in the exact same way as the 7 tests above.

testCheckEmailDomainExtensionValid()

The final edge case for the email that I decided to test was whether the domain extension was equal to .com or .ie. In reality there are probably other domain extensions that would also be valid but for the purpose of the question I took these as the only valid domain extensions. I constructed an instance of CheckedUser with a domain extension not equal to .com or .ie. This should throw an exception if the solution being tested has taken this edge case into account. I tested this in the exact same way as the 8 tests above.

checkToStringName()

This tests the toString method for an instance of CheckedUser constructed with just name. I constructed an instance of CheckedUser called testUser with the name "John Scully" and used assertEquals to assert whether testUser.toString() is equal to the string "CheckedUser: John Scully".

checkToStringNameAge()

This tests the toString method for an instance of CheckedUser constructed with name and age. I constructed an instance of CheckedUser called testUser with the name "John Scully" and age 22, and used assertEquals to assert whether testUser.toString() is equal to the string "CheckedUser: John Scully (22)".

checkToStringNameAgeEmail()

This tests the toString method for an instance of CheckedUser constructed with name, age and email. I constructed an instance of CheckedUser called testUser with the name "John Scully", age 22 and email John.Scully@ucd.ie", and used assertEquals to assert whether testUser.toString() is equal to the string "CheckedUser: John Scully (22), john.scully@ucd.ie".

grade.java:

I wanted to also be able to calculate the grade for a given solution to the exam question. To do this I made a class called grade which has all the same tests as TestCheckedUser (but with a few small changes which I will explain).

It has an instance variable total of type double. This refers to the total number of marks scored on the exam question. The total number of marks according to the question on Moodle is 25, so total is initialised to 25. Every time a test fails, a certain number of marks are docked, by decreasing the value of total. To do this I made a method called setTotal(), which takes in a value of type double (the amount we want to increase total by). this.total (ie the value of total for the current instance of grade) is increased by the amount passed in. For example, if total is equal to 25, setTotal(-0.5) will set total to 24.5. The method returns void. To then access this total variable I made a method called getTotal() which has no parameters and returns this.total (the current value of total for the current instance of grade).

I then had to decided how to divide up the marks.

Each part (a) - (e) had the number of marks outlined on Moodle. I then had to further divide these parts into the subparts and divide up the marks. I had no way to test part (e) so the minimum number of marks that a solution to the exam question can score is 5 marks.

The way I decided to divide up the marks is shown in the table below, which outlines the parts, subparts, tests to tests these subparts and the marks docked if these subparts are incorrect.

Part	Total Marks for this part	Subpart	Tests	Marks	Total Marks
(a) - accessor and mutator methods	3 marks	getName	checkGetName	0.5	
		getAge	checkGetAge	0.5	
		getEmail	checkGetEmail	0.25	
			checkGetEmailLowerCase	0.25	
		setName	checkSetName	0.5	
		setAge	checkSetAge	0.5	
		setEmail	checkSetEmail	0.5	3
(b) - constructors	3 marks	constructor for just name	checkNameConstructor	1	
		constructor for name, age	checkNameAgeConstructor	1	
		constructor for name, age, email	checkNameAgeEmailConstructor	1	3
(c)	12 marks	Name has 1 or 2 words	testCheckNameNull	1	
			testCheckNameLength	1	2
		no prefix	testCheckNamePrefix1	0.4	

Part	Total Marks for this part	Subpart	Tests	Marks	Total Marks
			testCheckNamePrefix2	0.4	
			testCheckNamePrefix3	0.4	
			testCheckNamePrefix4	0.4	
			testCheckNamePrefix5	0.4	2
		Age is between 18 and 30	testCheckAgeRange1	1	
			testCheckAgeRange2	1	2
		<name> equal to the instance variable name	testCheckEmailMatchesName1	1	
			testCheckEmailMatchesName2	1	2
		<somewhere>	testCheckEmailDomainSpecialCharacter	0.66	
			testCheckEmailDomainNull1	0.66	
			testCheckEmailDomainNull2	0.66	~2
		<something>	testCheckEmailDomainExtensionNull1	0.66	
			testCheckEmailDomainExtensionNull1	0.66	
			testCheckEmailDomainExtensionValid	0.66	~2
(d) - toString	2 marks		checkToStringName	0.66	
			checkToStringNameAge	0.66	
			checkToStringNameAgeEmail	0.66	~2
(e)	5 marks				5
			Total Marks		25

For the following tests:

checkGetName()
checkGetAge()
checkGetEmailLowerCase()
checkSetName()
checkSetAge()
checkSetEmail()

checkNameConstructor()
checkNameAgeConstructor()
checkNameAgeEmailConstructor()

the tests are the exact same as in TestCheckedUser except that within the catch block i call the method setTotal() and pass in the number of marks to be deducted if the test fails, which are outlined in the above table.

The tests **checkToStringName()**, **checkToStringNameAge()**, **checkToStringNameAgeEmail()** are the exact same as in TestCheckedUser except that I have now put the constructing the instance of a CheckedUser and the assertEquals in a try block, and then added a catch block to catch an AssertionError. Within the catch block i call the method setTotal() as above and pass in the number of marks to be deducted if the test fails, which are outlined in the above table.

In **every other test**, every thing is the same as in TestCheckedUser, except I have put the assertEquals in a try block, and then added a catch block to catch an AssertionError. Within the catch block i call the method setTotal() as above and pass in the number of marks to be deducted if the test fails, which are outlined in the above table.

The reason I put the AssertEquals in a try block and then made a catch block to catch an assertion error is so that the programme will deduct marks if the given solution to the exam question should fail for a certain test, but the programme will then pass the test and continue running. Therefore, the programme will not crash from tests failing and it will be able to continue and calculate the grade. However, it will therefore pass all tests (but deduct the marks for tests which should cause failure) and will not show which tests have been failed, which is why I have also included TestCheckedUser.

To calculate the grade I made the method calculateGrade which takes in a double which refers to the total marks, and returns a string - the grade.
It first converts the marks into a percentage and then returns the grade. I used the UCD Computer Science grading scheme which I found at the following link (<https://www.cs.ucd.ie/Grading/>).

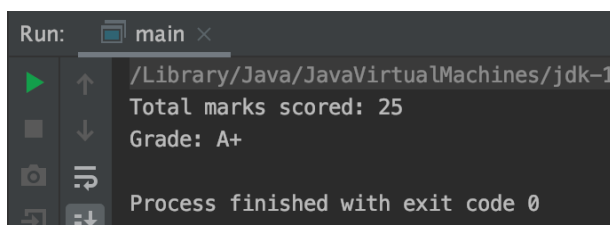
I made a method run all which contains all tests and will run all tests when it is called.

main.java:

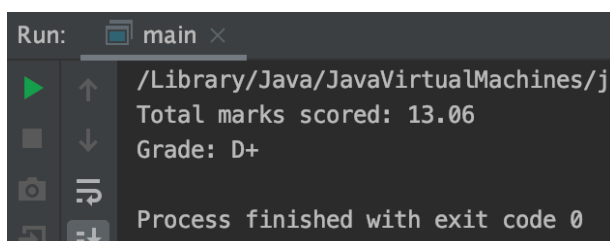
To calculate and print the grade for the CheckedUser, I made an instance of grade in main called solutionGrade (with empty constructor). I then ran all tests for instance of grade by solutionGrade.runAll(), so as that marks could be deducted appropriately. I then printed to the screen the grade by calling solutionGrade.calculateGrade() and passing in the total, which is solutionGrade.getTotal() - ie

System.out.println(solutionGrade.calculateGrade (solutionGrade.getTotal())).

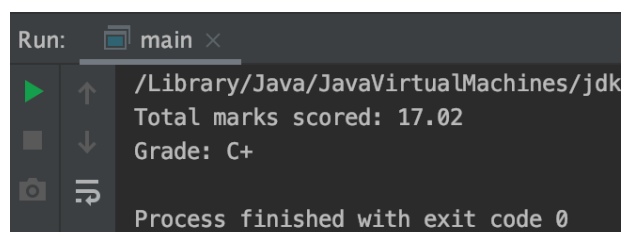
This prints the appropriate grade to the screen.



```
Run: main x
/Library/Java/JavaVirtualMachines/jdk-1
Total marks scored: 25
Grade: A+
Process finished with exit code 0
```



```
Run: main x
/Library/Java/JavaVirtualMachines/j
Total marks scored: 13.06
Grade: D+
Process finished with exit code 0
```



```
Run: main x
/Library/Java/JavaVirtualMachines/jdk
Total marks scored: 17.02
Grade: C+
Process finished with exit code 0
```