© Manfred Kerber
Alexandros Evangelidis

School of Computer Science
University of Birmingham

Worksheet 4

MSc/ICY Software Workshop

Assessed Exercise: 12.5% of the continuous assessment mark.

Submission: Thursday 19 November 2015 2pm

5% late submission penalty within the first 24 hours. No submission after 24 hours.
JUnit tests and JavaDoc comments are mandatory. All submissions must pass the tests provided on 11 November. Follow the submission guidelines on
`http://www.cs.bham.ac.uk/internal/courses/java/msc/submission.php`.

Exercise 1: (Basic, MSc: 30%, ICY: 35%)

(a) Write a class `ExamQuestion` with field variables `private String question` and `private int maximalMark`. Write a suitable constructor, getters/setters for the two field variables, and a `public String toString()` method which generates a String of the form `"Question (Maximal mark: <maximalMark>)\n" + "<question>\n"`, where `<maximalMark>` has to be replaced by the maximal mark and `"<question>"` by the question.

(b) Write a subclass `ExamQuestionSimpleChoice` of class `ExamQuestion`, in which the `possibleAnswers` are an `ArrayList<String>`, that is, the answer is supposed to be a choice from the list and the `correctAnswer` is of type `int`, representing the position of the correct answer (start counting from 1, since that is what humans normally do).

Write a method `public int answer(int value)` in the subclass that returns full marks if the answer is correct and `0` otherwise. Furthermore, write a suitable `public String toString()` method, making use of inheritance as far as possible.

For instance, if the answer to "2+3" is to be tested, it may be possible to offer the values 4, 5, 10, and 20 as possible answers to an `ArrayList<String>` with `ArrayList<String> a = new ArrayList<String>();` and `a.add("4"); a.add("5"); a.add("10"); a.add("20");` the right answer to the same question would be 2, (remember, we start counting the answers from 1 here). That is, with
`ExamQuestionSimpleChoice q1 =`
`    new ExamQuestionSimpleChoice("2+3 = ?", 10, a, 2);` we should get from `q1.answer(2)` the full `10` marks and from `q1.answer(3)` only 0 marks.

(c) Write a subclass `ExamQuestionNumeric` of class `ExamQuestion`, in which the answer is supposed to be a numerical answer of type `int` (again you have to represent the correct answer).

Write a method `public int answer(int value)` in the subclass that returns full marks if the answer is correct and `0` otherwise. Furthermore write a suitable `public String toString()` method, making use of inheritance as far as possible.

E.g. assume a question:
`ExamQuestionNumeric q2 = new ExamQuestionNumeric("2+3 = ?", 10, 5);`

where `10` is the maximal number of marks and `5` the correct answer.

`q2.answer(5)` should return `10`, whereas `q2.answer(6)` should return `0`.

<u>**Exercise 2:**</u> **(Basic, MSc: 20%, ICY: 25%)** Assume an abstract class `Student` with two subclasses `UGStudent` and `PGStudent`. All three classes have the field variables `String name` and `String registrationNumber`. Write an abstract method `passedSWS(double examination, double cA, double team)` of an appropriate type and write concrete methods in the two subclasses, which compute the final mark of a student on a module as a weighted average of 70% of the examination, 20% of the continuous assessment, and 10% of the team project mark. A UG student passes the module if their overall mark is equal to or greater than 40, an MSc student passes if it is equal to or greater than 50.

<u>**Exercise 3:**</u> **(Medium, MSc: 15%, ICY: 20%)** A manufacturing company produces several types of goods. Each good goes with an `orderCode`, a `price`, and its `availability` (of types `String`, `int`, and `boolean`, respectively). Some goods are `perishable`, they go in addition with a `bestBeforeDay` in form of a single `int` standing for the day of the year until they are fine. According to company policy such goods must not be shipped out after 14 days before the `bestBeforeDay`. For perishable goods you should write a method `public boolean sellable()` that checks whether the good can still be sold today. (Assume that a static method `int today()` is given that returns the day of the year for today. You may simulate this by a method that returns a value such as 311 for the 311th day of the year.) The customers of the company are to be represented in a `Customer` class, consisting of the field variables `name`, `address`, and `turnOver` of types `String`, `String`, and `int`, respectively. Some customers have "`goldStatus`", since they have a `turnOver` of more than £ 2000. They receive a 5% discount on all prices. Implement a method `public double toPay(int price)` which applies the discount to the price if appropriate.
How do you represent this best? Is it advisable to have a subclass `PerishableGood` for perishable goods? Likewise is it advisable to have a subclass `GoldStatus` for gold status customers? Justify your design decisions in appropriate comments and implement a corresponding program in appropriate classes.

<u>**Exercise 4:**</u> **(Advanced, MSc: 15%, ICY: 20%)** In the `BankAccountWithOverDraft` class from the lecture, we have seen how to override a method such as the `public void withdraw(long amount, String pd)` method. By overriding a method it can take on a completely different behaviour. For instance, it could mean for the withdraw method that it does not actually change the balance. Obviously this is unwanted and an implementer may want to enforce that a method cannot be overridden in any subclass. This is done by declaring it `final`, e.g., by `public final void withdraw(long amount, String pd)`. However, if we make just this change, it will not be possible to implement a subclass `BankAccountWithOverDraft`, since we cannot change the `withdraw` method any more.
What we do want is to change the code in the `BankAccount` class so that the `withdraw` method can be final (and thereby guarantee that the balance is correctly updated upon a withdrawal and that the password is checked for all subclasses of the class `BankAccount` as well). Still some code should be overridden so that it is possible to make use of the `overDraftLimit` in the `BankAccountWithOverDraft`, but not in the `BankAccount` class.

<u>**Exercise 5:**</u> **(Advanced, MSc: 20%, ICY: 0%)** Develop a test plan for the interactive `BankAccount` class as found on `http://www.cs.bham.ac.uk/internal/courses/java/msc/handouts/1-04/BankAccounts.java` and its three accompanying classes `BankAccount`, `Transaction`, and `Customer`. Your test plan – to be included in your zip file – should consist of two A4 pages of point size 11 font and be submitted in accessible PDF format.