MSO 2016–2017
Solutions
October 6, 2016, 9:00 - 11:00

Name:

Student number:

Please read the following instructions carefully:

- Fill in your name and student number above. Be prepared to identify yourself with your student card when you submit your exam.
- All answers should be filled in on these sheets. The other answer sheets can be used for initial sketches (kladpapier).
- This is a closed-book exam. You are forbidden from accessing any external material, notes, electronic material, or online resources.
- Answer each open question in the space provided. Write clearly and legibly. Answer each multiple choice question by marking the space provided.
- A maximum of 100 points can be obtained by the questions of this exam, to be divided by 10 and incremented to yield the overall mark for the exam.
- Participants who claim extra time may hand in their work until 11:30.

Please do not write in the space below.

Question	Points	Score
Multiple choice	28	
Requirements	20	
Domain models and GRASP	24	
Design patterns	28	
Total:	100	

Question 1: Multiple choice (28 points)

(a) (4 points) *Long methods* containing more than 200 lines of code, may indicate:

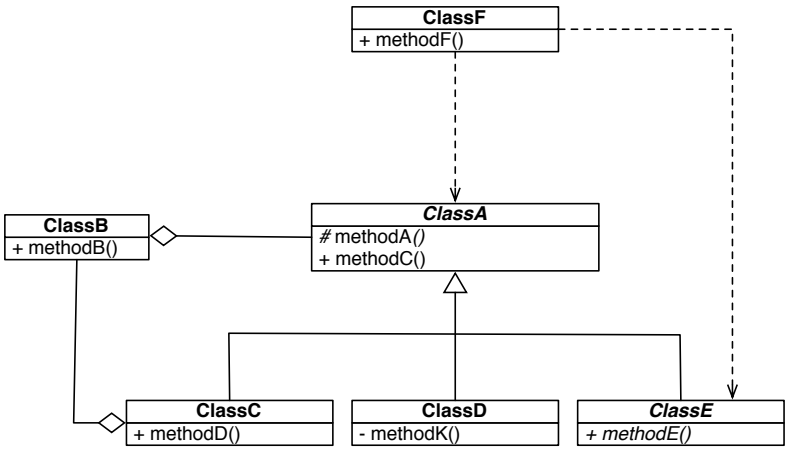
- ◯ Strong cohesion – by grouping you code in long methods, your classes become more compact and strongly cohesive;
- √ **Weak cohesion – this method may be doing too much or too many different things;**
- ◯ Loose coupling – by organizing code in longer methods, you have fewer methods to implement, which are coupled more loosely;
- ◯ Tight coupling – the method is tightly coupled to its implementation.
- ◯ All of the above;

(b) (4 points) What is the desired outcome of the *Elaboration* phase of the Unified Process?

- ◯ A fully functional software system that has been deployed and tested;
- ◯ A vision, scope, and business case that is shared by the different stakeholders;
- ◯ A full and careful design of the complete software system;
- √ **A core architecture, requirements document, and overall assessment of schedule and risks;**

Choose *one* answer.

(c) (20 points) Given the UML diagram below, which of the following statements is true? Multiple answers may be true.



- √ **ClassD is a concrete subtype of the abstract class ClassA;**
- ◯ ClassF has attributes of both ClassA and ClassE;
- ◯ Because ClassA is abstract, it cannot implement any concrete methods;
- √ **If you're implementing methodK, you can call methodA;**
- ◯ If you're implementing methodB, you can call methodD;
- √ **If you're implementing methodK, you can call methodC;**
- √ **If you're overriding methodA in ClassC, you can call methodC;**
- √ **methodB is public;**
- ◯ Using *polymorphism*, we can provide an object of ClassA, whenever a method requires an object of ClassD.
- ◯ Because the diagram contains a cycle, this design cannot be implemented.
- ◯ methodF cannot call methodE because it is abstract.

Question 2: Requirements (20 points)

Below are several requirements for an online gaming environment.

The requirements listed below are poorly formulated for a variety of reasons. *For each requirement, give **both** a brief explanation of what is wrong with the current formulation and suggest a better alternative.*

You have quite some creative freedom when coming up with exact requirements. Here is one example to get you started.

The server should be able to host large images.

This requirement could be phrased better as:

The server should be able to host images of at least 10 megabytes.

Here the precise limit (10MB) is arbitrary and left to your discretion.

(a) (4 points) There should be appropriate user identification.

> **Solution:** This requirement is not specific enough. How is user identification realized? By username/password? By other methods? Is user identification related to specific privileges?

(b) (4 points) There will be several classes of users with different privileges.

> **Solution:** This requirement is not specific enough. How are the classes identified/characterized? Which are the privileges?

(c) (4 points) There is no upper limit to the number of subscribers.

> **Solution:** This requirement is not specific enough. Subscription should be realized in such a way that there is no hard limit. One should specify concrete figures, e.g. 100.000 or 10 million.

(d) (4 points) There is no upper limit to the number of online players.

> **Solution:** This requirement is not specific enough. The system should be flexible enough to deal with high (specify in more detail) numbers of users, for instance by expanding the computing power in a cloud environment.

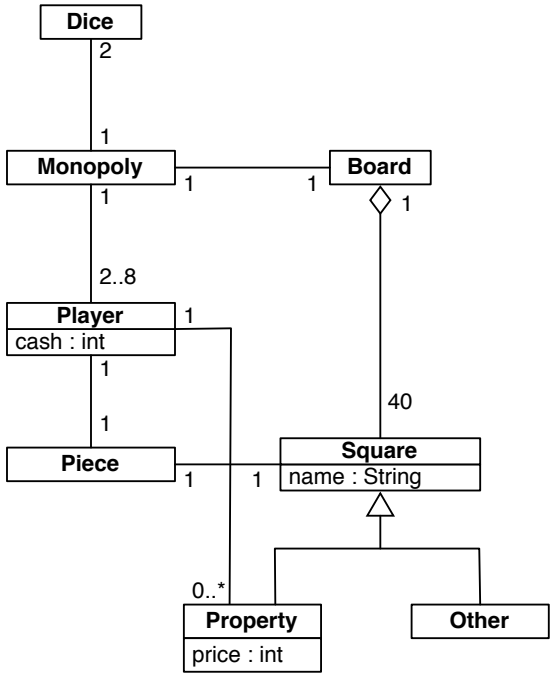
(e) (4 points) Online support will be offered.

> **Solution:** This requirement is not specific enough. What kind of support is offered? Just a short getting started? A long manual? Some kind of intelligent tutor? A users forum?

Question 3: Domain models and GRASP (24 points)

(a) (12 points) Give a domain model for the monopoly game.

Solution:



(Different correct answers may exist)

(b) (4 points) Which class should be responsible for creating the squares? Motivate your choice using GRASP principles.

Solution: According to the creator pattern, the Board should create the squares. Alternatively, a special Controller should be responsible for the board initialization.
(Different correct answers may exist)

(c) (4 points) Which class should be responsible for deciding when a player goes to jail? Motivate your choice using GRASP principles.

Solution:
The information expert suggests that the Monopoly game, which knows about both the dice and the player, should be responsible. Alternatively, a specific controller could handle the dice rolls. (Different correct answers may exist.)

(d) (4 points) Which class should handle the purchasing of property? Motivate your choice using GRASP principles.

Solution: The Player has all the information necessary: it knows (by means of the piece), which square he/she is on, and that square's price (if it is property). Hence by the Expert principle, it should be the Player. Alternatively, a specific controller could handle the purchasing of property. (Different correct answers may exist.)

Question 4: Design patterns (28 points)

(a) (6 points) Give the three basic principles for object oriented design patterns.

Solution: Design to an interface, not to an implementation. Favour aggregation over inheritance. Find what varies and encapsulate it. (3 x 2 points)

(b) (6 points) Give a concise description of the differences between the Facade and the Adapter pattern.

Solution: Facade: create your own interface - Adapter: adapt to a given interface.
Facade simplifies - Adapter does not.
Adapter probably uses polymorphism - Facade probably not.
(3 points per correct answer)

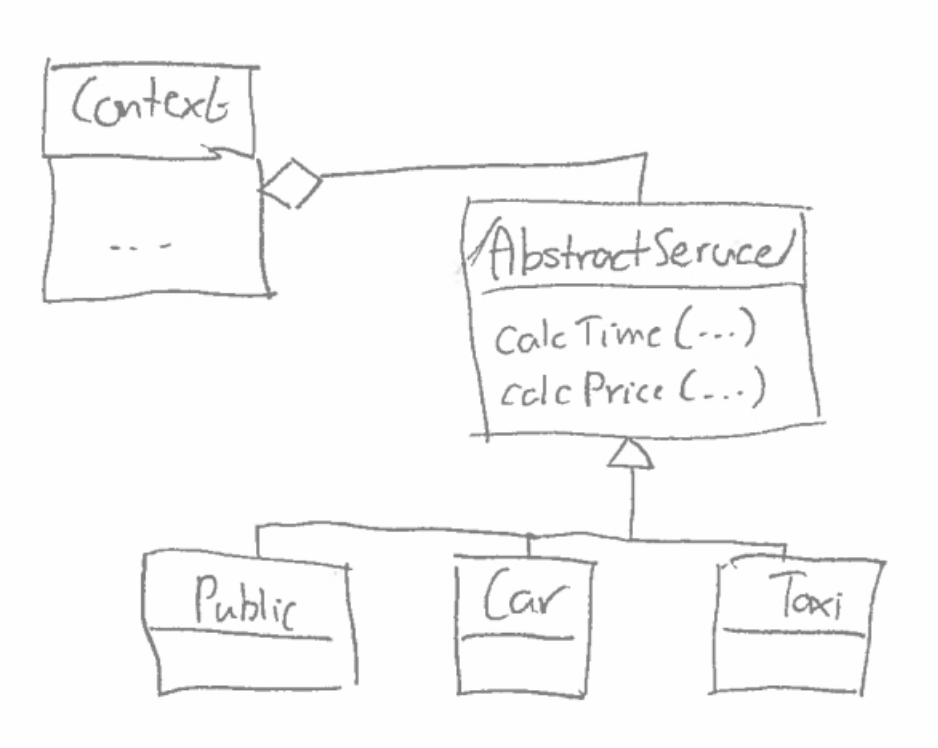
(c) (16 points) For a specific airport information system, we have a class UserInfo. This class provides two services with respect to travelling to the airport. There are three ways of travelling: public transportation (train, tram, bus), private car and taxi. One service is calcTime(...), which gives an estimation of the travelling time to the airport. The other service is calcPrice(...), which gives an estimation of the price.

The UserInfo class has information about the desired arrival date and time, the starting place (from an address database) and the way of traveling.

Which design pattern would you apply in this case? Draw a class diagram. Which parameters should the two methods have?

Solution: This is a candidate for Strategy. We encapsulate the variation in algorithms. (4 points) So the way of travelling will be used to create the right concrete strategy object. The two methods should be called with date+time and starting place as parameters. (4 points)

You may also use the next page for answering these questions.


Context
...
AbstractService
calcTime(...)
calcPrice(...)
Public
Car
Taxi