PA1434 Objektorienterad Design

First Resit

Mikael Svahnberg (0455-385811), Ludwik Kuzniarz (0455-385853)

2016-08-19

Points

(Filled in by the Marker)

Question:	1	2	3	4	5	6	7	8	9	10	11	SUM
Max Points:	4	4	4	4	4	4	6	4	4	4	4	46
Points:												

Grade:	

Instructions

- **Please Remember** to provide an answer in *each* checkbox [ ].
- **Marking** All multiple choice questions give four points. Each wrong answer in a question subtracts one point, but never below zero.
- **Diagrams** In some questions we ask you to draw a diagram (for example a class diagram or an interaction diagram). Please use a separate paper to make a draft first, and then redraw them in the marked area on the exam paper. Try to arrange the elements (and especially connecting lines) so that it is easy to read.
- **Allowed Books**: English to Swedish Dictionary.
- **Allowed Material:** Pen, Eraser, Ruler, Candy.

Good Luck!

Question 1. Interaction Diagrams 4P

Please mark *each* of the following statements as true **T** or false **F**:

- [ ] A *sequence diagram* and a *communication diagram* present the same things.
- [ ] A *system sequence diagram* is just a special case of a sequence diagram, where the system is viewed as a black box.
- [ ] An interaction diagram contains the state of all attributes in all objects.
- [ ] The entity `:Volvo` is an object without a name.
- [ ] The entity `volvo:Car` is a class of the type `Car`.
- [ ] You can always use a *State diagram* instead of an *Interaction diagram*.

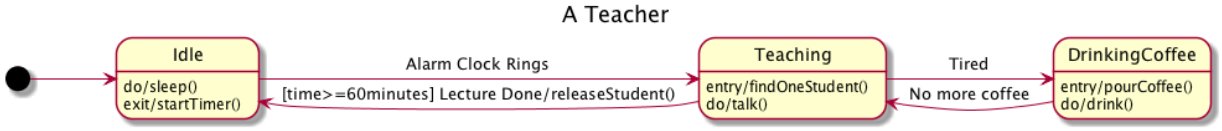
Question 2. Class Diagrams 4P

Please mark *each* of the following statements as true **T** or false **F**:

- [ ] A conceptual model uses the same notation as a class diagram.
- [ ] An association between two classes means that one of the classes must have at least a pointer to the other class.
- [ ] Class diagrams may be nested inside a *Package diagram*.
- [ ] In `[Class A]` ≪— `[Class B]` , the ≪— means that class `B` inherits from class `A`.
- [ ] In `«Singleton» PrinterDriver`, the `«Singleton»` is a stereotype that indicates that there will always be exactly one object of the type `PrinterDriver`.
- [ ] The visibility of methods in a class can only be *public* or *private*.

Question 3. State Machine Diagrams 4P

Please study the following diagram, and then assess each of the claims below:

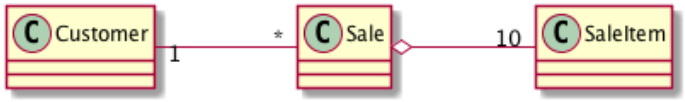


Claims: (Please mark *each* claim as true **T** or false **F**)

- [ ] The `releaseStudent()` is an action that is called when the teacher is tired.
- [ ] The students will have to watch the teacher while they drink their coffee.
- [ ] Regardless of how many times the teacher drinks coffee, they will always only teach for one student at the time.
- [ ] If the lecture is done in 30 minutes, the teacher can go back to sleep early.
- [ ] The teacher may continue talking while drinking their coffee.
- [ ] If the lecture is done, the time is 50 minutes, and the teacher is tired, he must get more coffee.

Question 4. Relations between Classes 4P

Please study the following diagram, and then assess each of the claims below:



Claims: (Please mark *each* claim as true **T** or false **F**)

- [ ] A `Sale` has precicely one customer associated with it.
- [ ] A `Customer` may have any number of sales.
- [ ] `si:SalesItem` may be moved from `sale1:Sale` to `sale2:Sale`.
- [ ] The association between `Customer` and `Sale` is best implemented as an array in `Customer`.
- [ ] There may only be 10 `SaleItems` in this system
- [ ] A `Sale` can consist of up to 10 `SaleItems`

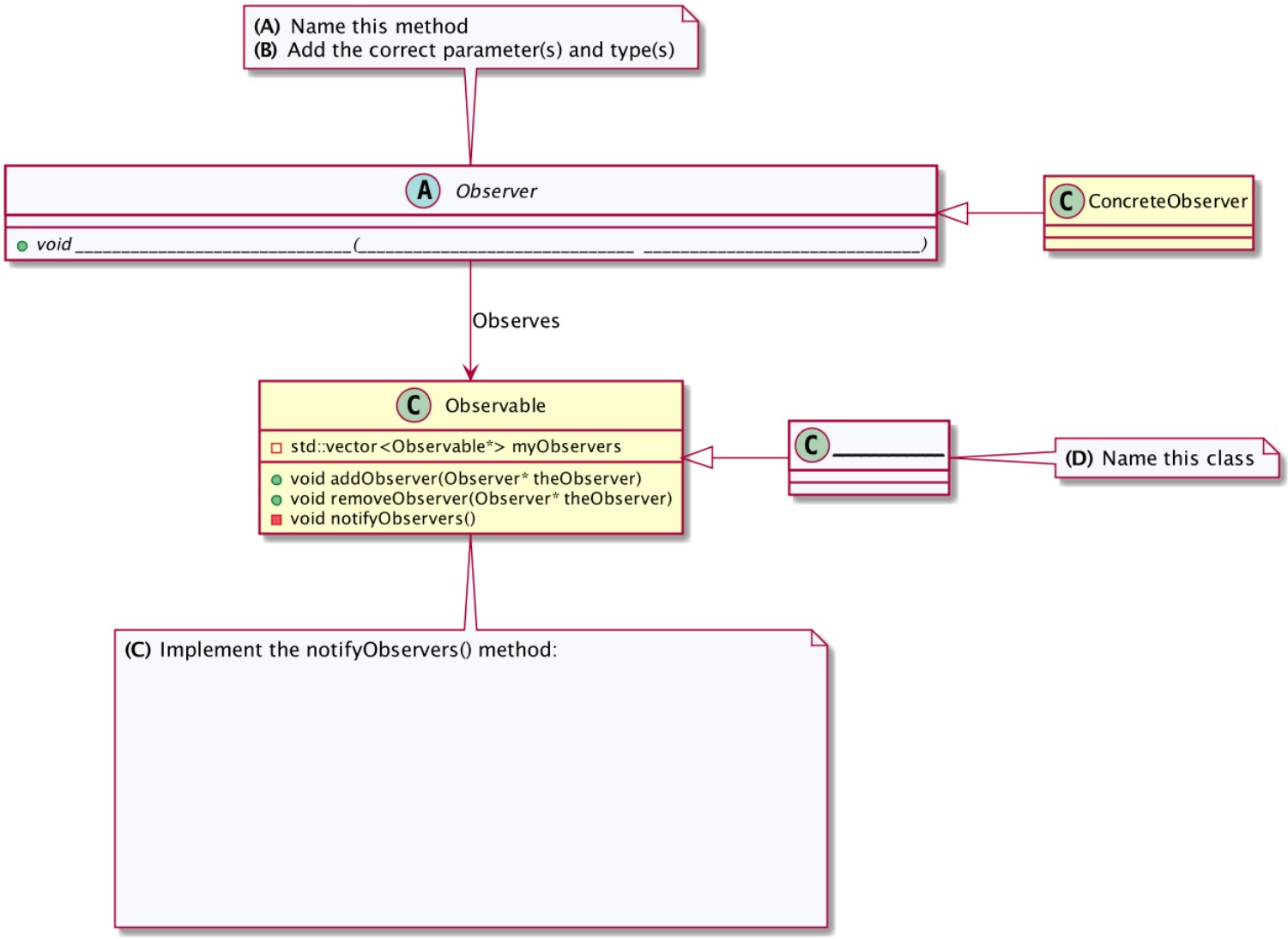
Question 5. GRASP Patterns 4P

Please mark *each* of the following statements as true **T** or false **F**:

- [ ] A *Controller* is a class that accepts system events and delegates to other classes to perform the requested function.
- [ ] An *Information Expert* is responsible for doing something because it has all the necessary information to do so.
- [ ] The *Low Coupling* pattern suggests to create fewer dependencies between classes.
- [ ] The *High Cohesion* pattern means that each class should have as many responsibilities as possible.
- [ ] A *Creator* is a system-wide class that is responsible for creating new instances of all other classes.
- [ ] A *Controller* is responsible for controlling that the result of an operation is correct.

Question 6. Design Patterns 4P

The following class diagram illustrates the *Observer* design pattern. Please add the four missing elements (A to D) in the diagram.



Case Description for the Remaining Questions

The remaining questions uses the following case. Each paragraph is numbered for easy reference in the questions. Please read this description carefully before answering the questions below.

Simplified Monopoly

(1) *Simplified Monopoly* is a board game where you have

- a board,
- up to five players,
- two dice,
- houses,
- hotels,
- money,
- 32 cards, with 16 cards each of *chance* and *community chest* cards, and
- deeds for each of the following properties:
    - 22 streets
    - 4 railroads
    - 2 utilities

(2) Each player has a name, a token (battleship, racecar, thimble, boot, or a top hat), and some money. A player can buy and own any number of properties. On a property a player can build houses and hotels.

(3) One player acts as the bank, and is responsible for all the unsold properties and all the money not owned by a player. This player is also responsible for interpreting the rules of the game, a so-called *game master*.

(4) Players take turn to roll dice and move their token accordingly (in this game, the player has to manually move the token). A token has a position on the board. Players may have to take a card, and has to act according to what the card says.

(5) When a player lands on a property that is owned by another player, they have to pay rent. In this version of the game this is constructed as a debt from one player to another, and the player can decide to pay the debt immediately or pay an interest rate (agreed between the two players) per lap.

(6) Every time a player completes a lap, they collect £ 200 from the bank.

(7) There is no end to the game. There is just poverty and despair.

Question 7. Conceptual Model **6P**

Please construct a *Conceptual Model* for this system. If and where applicable, use associations, aggregation, composition, inheritance, and association attributes. Please also consider multiplicity where stated and applicable.

Question 8. Use Case Diagram 4P

Please draw a *Use Case Diagram* for the *Simplified Monopoly* system. Make sure you identify all *Actors* as well as all *Use Cases*. If applicable, you may structure your use case diagram so that some use cases *include* or *extends* other use cases.

Question 9. Expanded Use Case 4P

Please write an *Expanded Use Case* for the `Roll Dice and Move` Use Case. This use case covers the case when it is a players turn to roll the dice, move their token, possibly pay rent, possibly take a card, and possibly collect money from the bank (paragraphs 4 to 6 in the case description above).

Use Case: `Roll Dice and Move`

Primary Actor:

Actors:

Preconditions:

Postcondition:

Brief Description:

Continued on the next page

Basic Flow:

Actor	System

Alternative Flows:

Question 10. Interaction Diagram **4P**

Please draw an *Interaction Diagram* (Sequence or Communication Diagram) for the `RegisterDebt(fromPlayer, toPlayer, amount, interestRate)` system operation.

Question 11. Class Diagram 4P

Please draw a partial class diagram that includes the classes, methods, and attributes that you used in the interaction diagram in the previous question. Remember to also consider the associations between classes, and add (if and where applicable) base classes and inheritance hierarchies.