
MSO 2012–2013
EXAM
January 30 2013, 13:30 – 16:30

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.
 - 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 102 points can be obtained by the questions of this exam, to be divided by 10 to yield the mark for the exam. There is one bonus question about the guest lectures at the end of the exam.
 - After the exam a version with possible answers will be made available.
-

Please do not write in the space below.

Question	Points	Score
Multiple Choice	21	
Sudoku	20	
Social media	20	
Decorator	20	
Creational patterns	18	
Closing	3	
Total:	102	

Question 1: Multiple Choice (21 points)

- (a) (3 points) Which of the following statements is *incorrect*?
- The Unified Process plans short iterations to make the software development process more robust against changes in requirements.
 - The Unified Process consists of various phases. Each phase consists of a different mix of the Analysis, Design, Implementation, and Testing phases of the waterfall model.
 - The phases of the Unified Process mix the Inception, Elaboration, Construction, and Transition phases. As the iterations progress, the balance between these phases shifts.
 - Each iteration aims to deliver a 'base line product' that could in principle be released. The subsequent iterations improve upon this baseline.
- (b) (3 points) What is the purpose of use cases?
- To establish the user interface of a software system.
 - To establish the design patterns necessary to implement a software system.
 - To establish the functional requirements of a software system.
 - To establish both the functional and non-functional requirements of a software system.
- (c) (3 points) How does the Strategy pattern adhere to the Open Closed Principle?
- The Strategy pattern exposes an Open interface to client code, but hides concrete strategies behind a Closed abstract class.
 - The Strategy pattern provides Open public methods of a Closed abstract class.
 - The Strategy pattern Opens a loosely coupled abstract class, but Closes the strongly cohesive concrete classes.
 - The Strategy pattern is Open for extension by introducing new concrete strategy classes, but each concrete Strategy class is Closed for modification.
- (d) (3 points) What happens when you break the Liskov Substitution Principle?
- This may result in unexpected behaviour when using instances of the subclasses involved.
 - You cannot use *polymorphism* to use a subclass when a method expects an argument of its parent class.
 - The information *encapsulated* by an abstract class may become visible to the class's clients.
 - You fail to program to an *interface*, instead of an *implementation*.
- (e) (3 points) Which of the following is *not* a *drawback* of the Singleton Pattern?
- It relies on static methods, that are not supported by all programming languages.
 - It limits the number of instances that can be created.
 - It is not threadsafe.
 - It can create implicit coupling between the Singleton's clients.
- (f) (3 points) Why would you apply the Object Pool Pattern?
- Because you want to control the number of instances of a certain class.
 - Because you want to control access to the information in a certain class.
 - Because you want to control the number of clients using a certain class.
 - Because you forgot your swimming trunks.
- (g) (3 points) Why would you apply the Facade Pattern?
- Because you need to introduce a simple interface to a complex subsystem.
 - Because you need to adapt the interface of an existing system to fit with legacy code.
 - Because a system needs to run on different operating systems or platforms.
 - All of the above.

Question 2: Sudoku (20 points)

Sudoku and Kakuro are two kinds popular puzzle formats.

1								6
		6		2		7		
7	8	9	4	5		1		3
			8		7			4
				3				
	9				4	2		1
3	1	2	9	7				4
	4			1	2		7	8
9		8						

An example Sudoku puzzle

	23	30				27	12	18		
16					24					
	17				17					
				29						
35				16						
							12			
			7			8				
						7			7	
					18					
		11	10							
								5		
21										
6								3		

An example Kakuro puzzle

A Sudoku puzzle consists of a 9x9 grid of squares, where some squares have been filled in with a digit between 1 and 9. The aim of Sudoku is to fill in the remaining squares with a digit between 1 and 9 in such a way that:

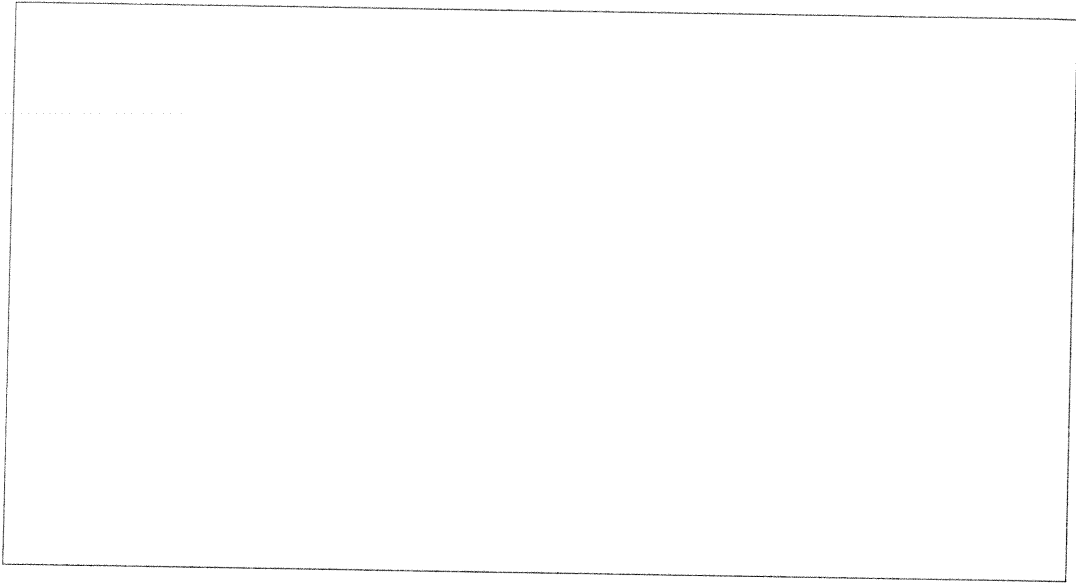
- all nine rows of the Sudoku puzzle contain the numbers from 1 to 9 without duplicates;
- and all nine columns of the Sudoku puzzle contain the numbers from 1 to 9 without duplicates;
- and the nine 3x3 squares of the Sudoku puzzle contain the numbers from 1 to 9 without duplicates.

Kakuro puzzles also consist of a grid of squares. Usually the grid consists of 16x16 squares, although this may differ. The squares in Kakuro may be black or white. Apart from the top row and leftmost column which are entirely black, the grid is divided into *entries* – lines of white cells – by the black cells. The black cells contain a diagonal slash from upper-left to lower-right. Every black cell has a number in none, one, or both halves. Each horizontal entry has a number in the black half-cell to its immediate left and each vertical entry has a number in the black half-cell immediately above it. These numbers, borrowing crossword terminology, are commonly called *clues*. The object of the puzzle is to insert a digit from 1 to 9 inclusive into each white cell such that:

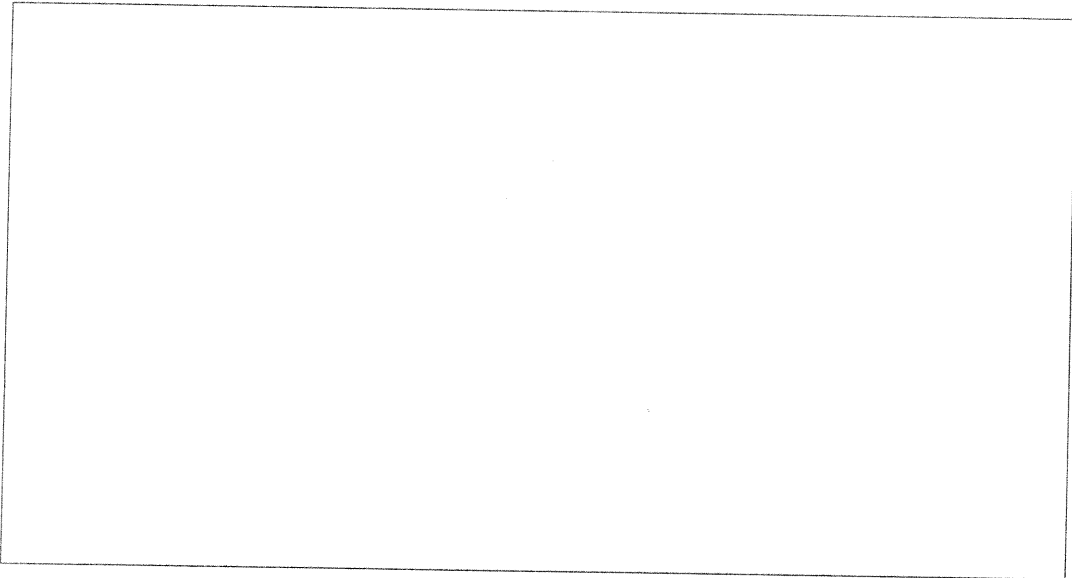
- the sum of the numbers in each entry matches the clue associated with it;
- and that no digit is duplicated in any entry.

Now suppose you are asked to implement a computer game that lets you play both Sudoku and Kakuro. Given an incomplete puzzle, the user must fill in the squares one by one. At every step, the user's move is checked to be valid.

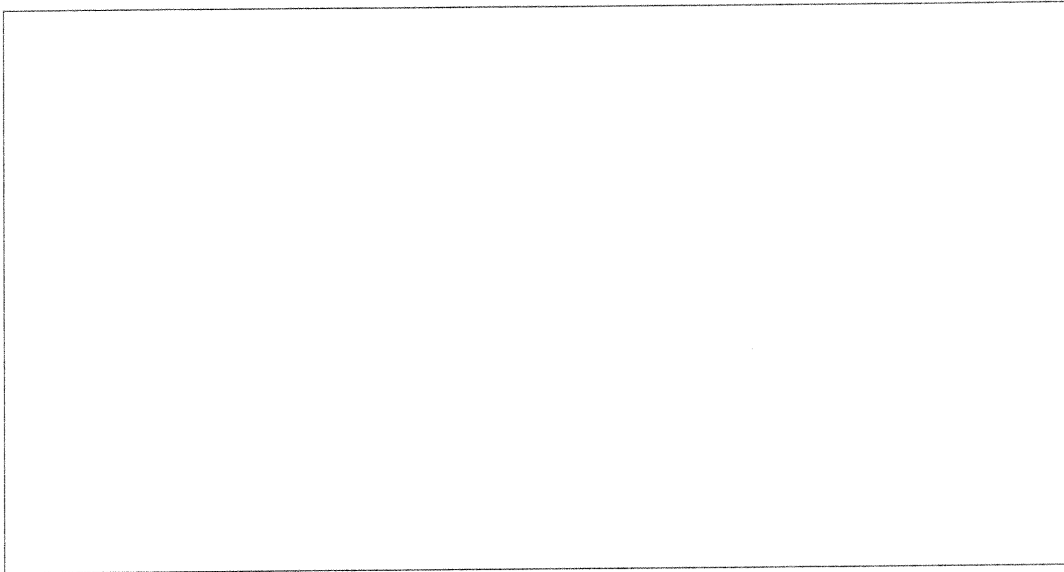
- (a) (4 points) Give a domain model for these two puzzles.



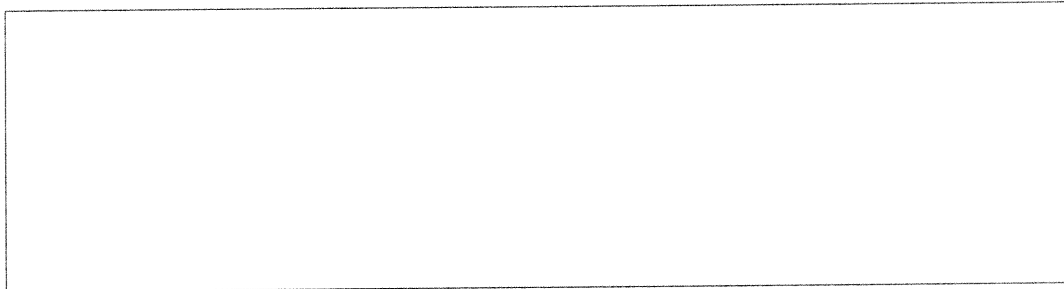
- (b) (4 points) Perform a Commonality-Variability Analysis (CVA) of Sudoku and Kakuro.




- (c) (4 points) Give an Analysis matrix of the Kakuro and Sudoku games.



- (d) (3 points) How would you assign responsibility for checking the validity of a move? Motivate your choice using the GRASP principles.



- (e) (5 points) Which design pattern would you use to check whether a Kakuro or Sudoku move is valid? Briefly motivate your choice and sketch a UML class diagram illustrating your design.



Question 3: Social media (20 points)

Social media sites, such as Facebook, Twitter, and Google+, allow users to post status updates. Suppose that you and your friends have been working on a new social media site. Users are allowed to post short status updates that other users can read in their news feed. This basic functionality has already been implemented.

As your site grows, your users have expressed interest in defining custom views on their news feed. This new functionality lets users filter and sort their newsfeed in different manners. Examples could include:

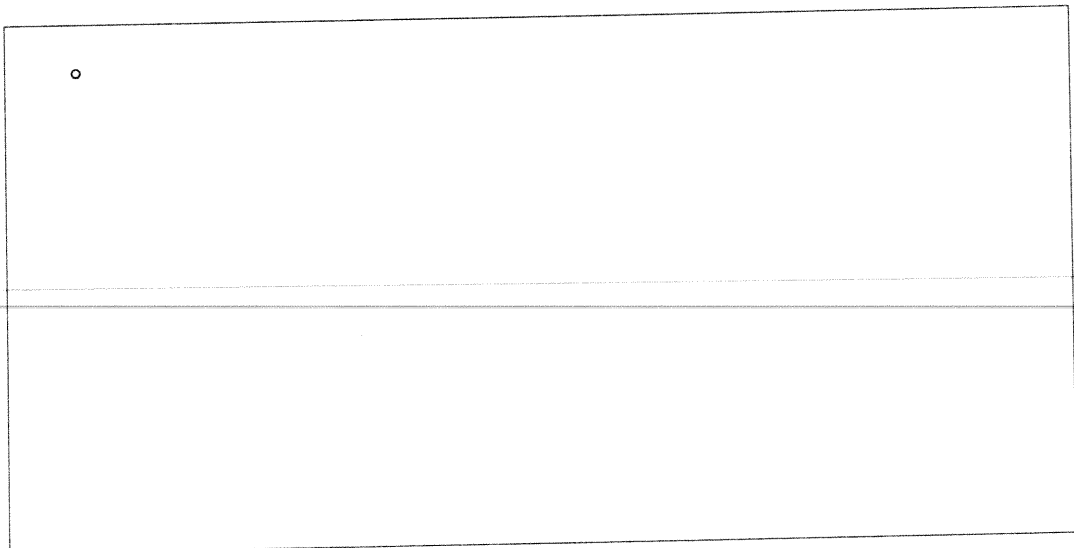
- a news feed of updates tagged with #MSO sorted by popularity;
- a news feed of updates sorted by retweet count in the last week;
- a news feed of nearby updates sorted by date;

The existing code that handles the printing of news feeds is currently structured as follows:

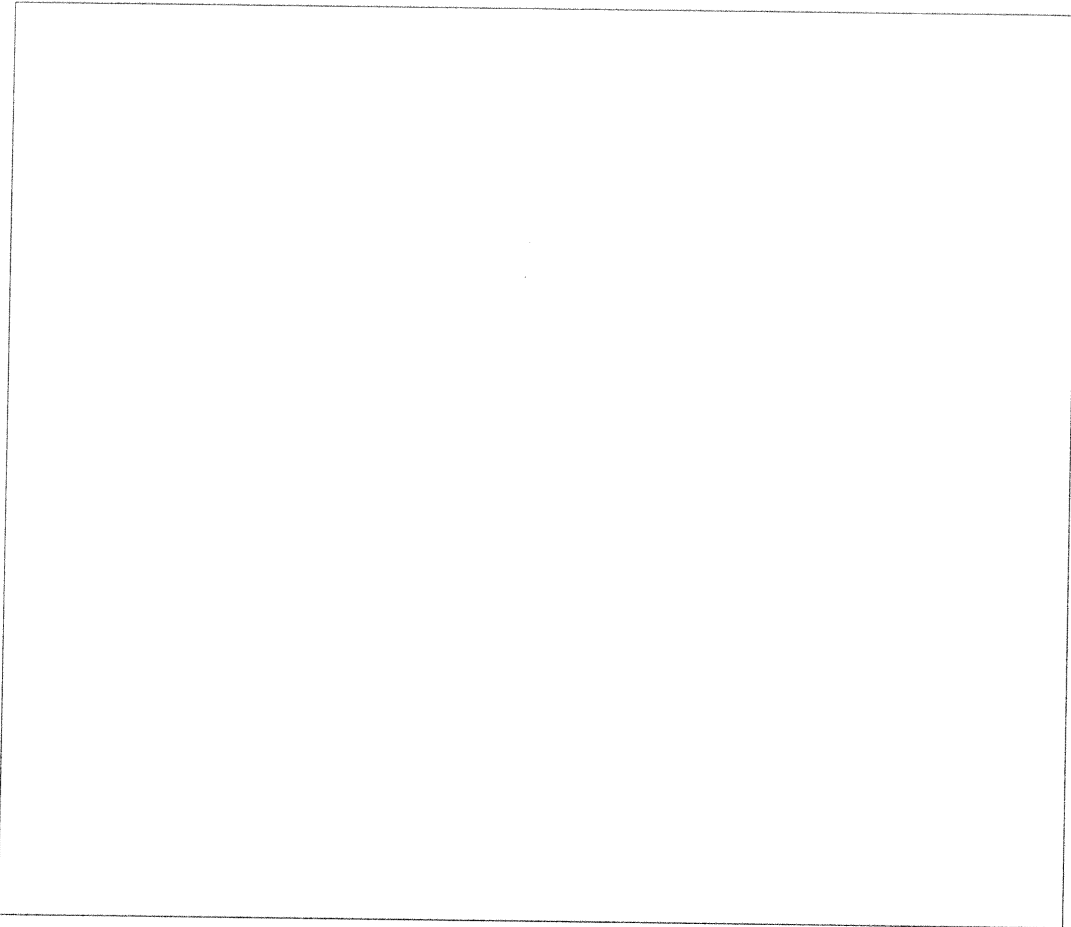
```
class NewsFeedCreator
{
    List<Update> makeNewsFeed(User user)
    {
        List<Update> newsfeed = new List<Update>();
        List<User> friends = user.getFriends();
        foreach (User friend in friends)
        {
            addUpdates(newsFeed, friend.getUpdates());
        }
        sortByDate(newsFeed);
        return newsFeed;
    }
}
```

In this question you will try to extend this class with the desired new functionality.

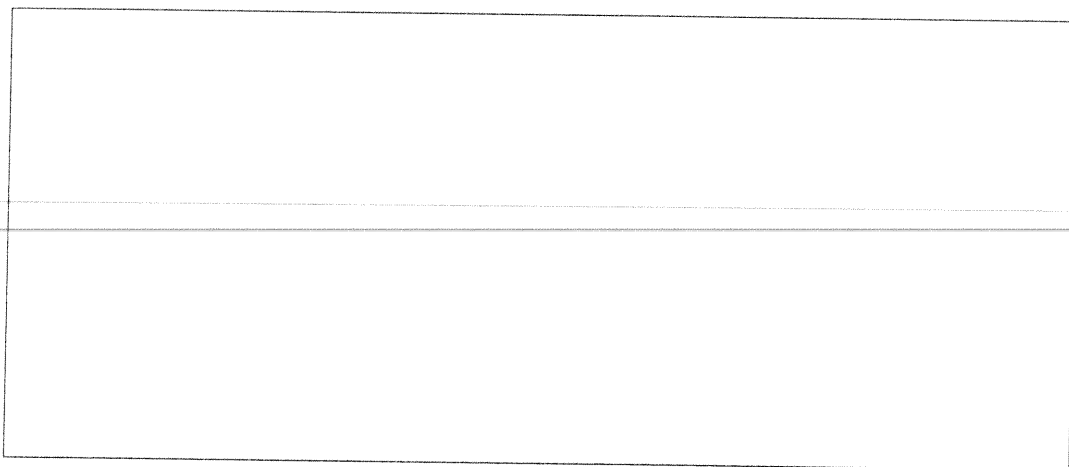
- (a) (4 points) Give a UML class diagram sketching a possible design applying the Strategy Pattern to handle both sorting and filtering.



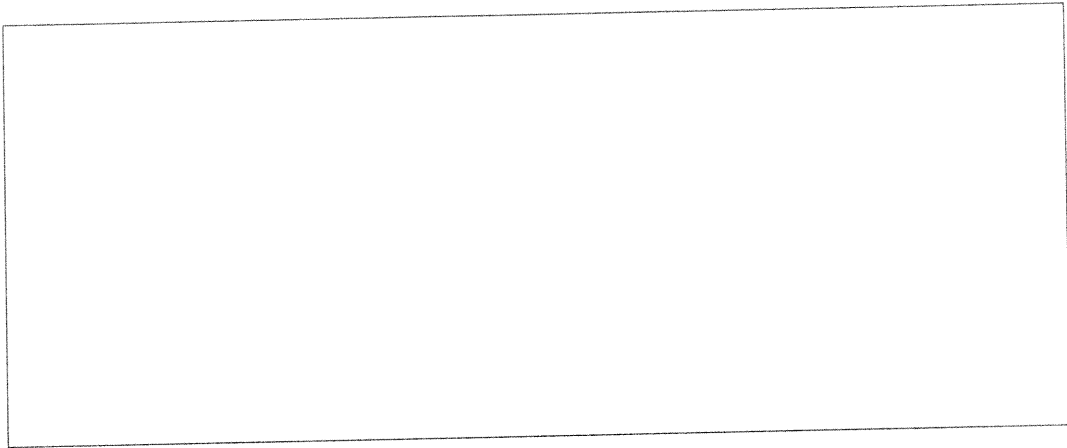
- (b) (6 points) Give a UML class diagram sketching a possible design using the Template Method Pattern. Also provide the code of the Template Method itself.



- (c) (5 points) In practice, almost all views chronologically sort the newsfeed. How could you take this information into account in your design using the Template Method Pattern?



- (d) (5 points) Which design would you prefer: the design using the Strategy Pattern or the design using the Template Method Pattern? Briefly motivate your choice.

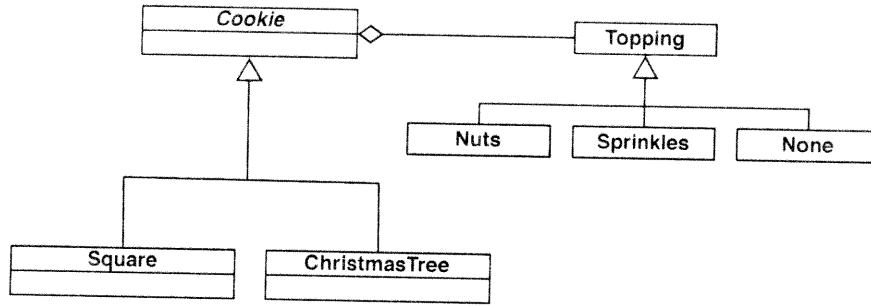


Question 4: Decorator (20 points)

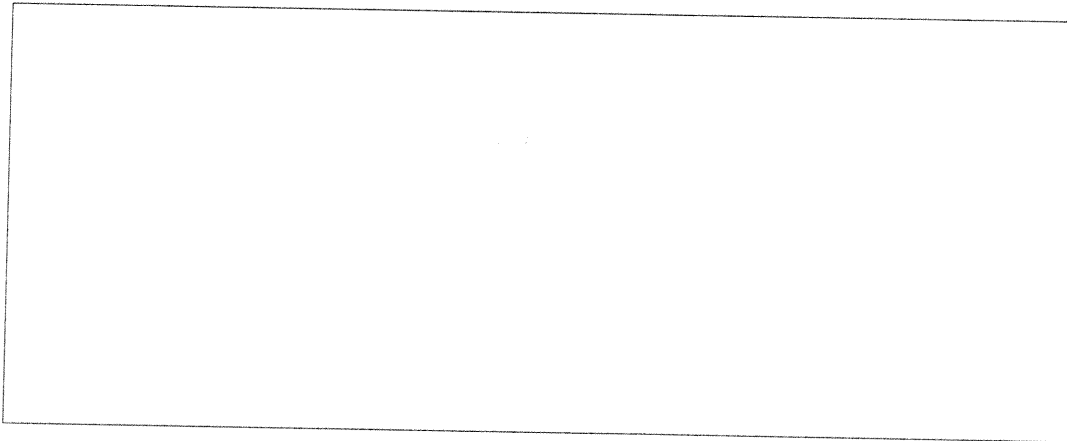
You have been asked to design a new version of CookieCutterPro, the latest version of a software package to automate the production of a variety of baked goods.

CookieCutterPro can produce cookies of various shapes: simple squares and circles, but also seasonal shapes, like christmas trees and easter bunnies. Cookies may be decorated with multiple toppings, including sprinkles, nuts, or chocolate.

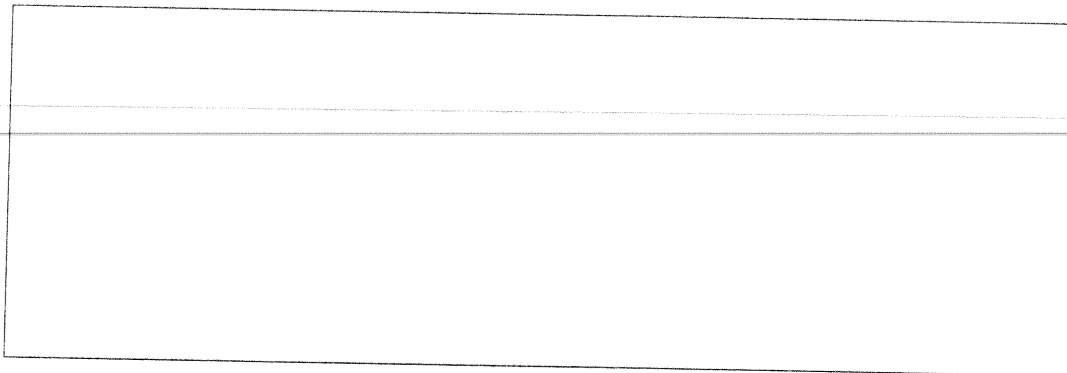
After this discussion, your fellow designer Andrea presents the following design(fragment):



- (a) (4 points) Another colleague, Bertrand, suggests that you should use the Decorator pattern. Give a UML class diagram how to use the Decorator pattern in this example.



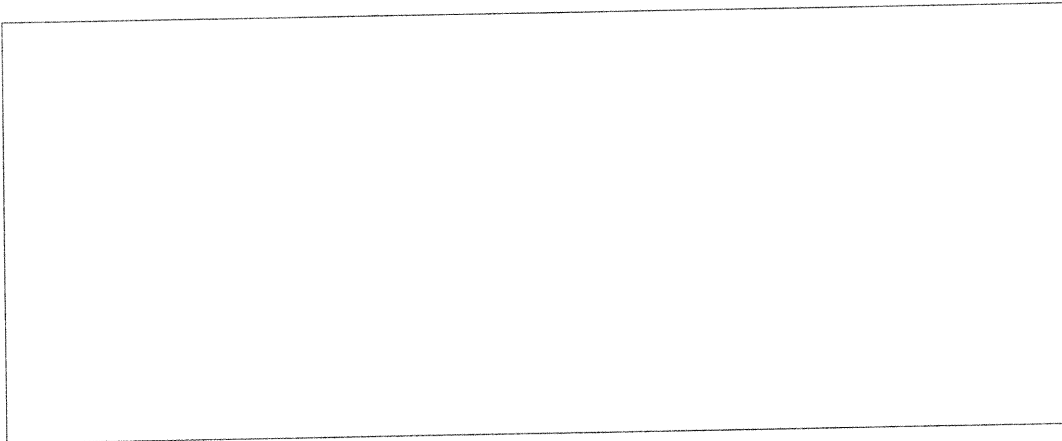
- (b) (4 points) For what reasons would you prefer to use the Decorator pattern over Andrea's solution?



- (c) (4 points) Now suppose that we need to produce a series of baking instructions for every cookie. For example, to produce a cookie that has both a chocolate sauce and nut topping, we would like to see the following instructions:

```
Bake cookie.  
Dip in chocolate.  
Sprinkle with nuts.
```

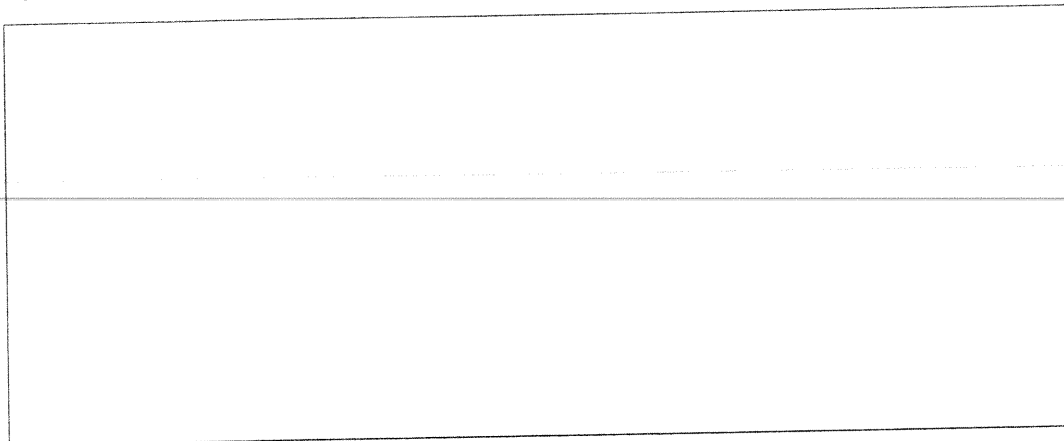
Give an implementation of the methods for computing this recipe for the Square and the Nuts classes. You may use a method `emitInstruction(String i)` that prints the argument string to the recipe being constructed.



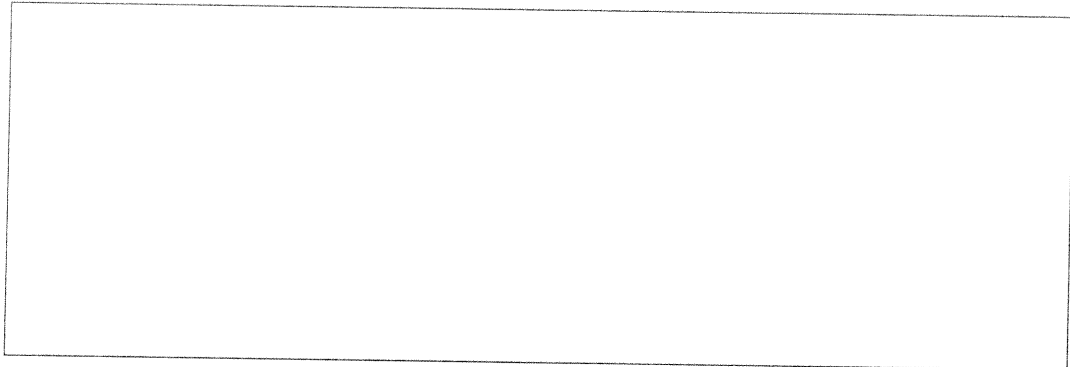
- (d) (4 points) Besides toppings, you may also want to incorporate extra ingredients in the cookie dough. These should be mixed in *before* the cookie is baked. For example, to make chocolate chip cookies, sprinkled with nuts we might want to see a recipe like this:

```
Mix in chocolate chips.  
Bake cookie.  
Sprinkle with nuts.
```

Explain how the Decorator Pattern can be used to add extra ingredients to the cookie's dough *before* it is baked:

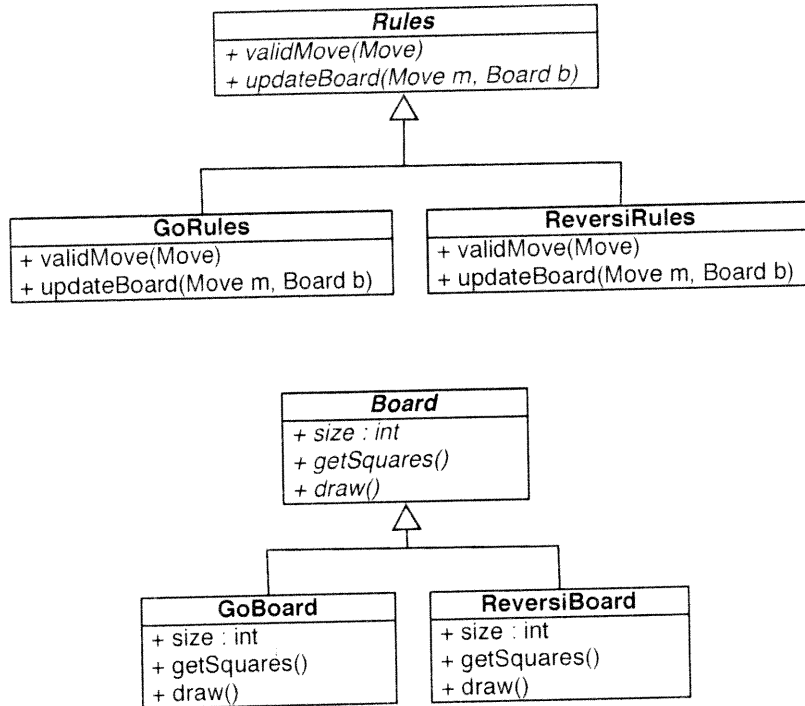


(e) (4 points) What kind of change in requirements would be hard to cope with in this design?



Question 5: Creational patterns (18 points)

Suppose you are implementing a computer game that lets you play the boardgames Go and Reversi. During your design you have the following situation:



- (a) (6 points) Explain how the Abstract Factory could be used to ensure that you do not end up playing Go on a Reversi board or visa versa. Sketch how to extend the design above, applying the Abstract Factory Pattern.

- (b) (4 points) Suppose another developer, Charles, has written the board and rules associated with TicTacToe (boter-kaas-en-eieren). Here is are two key classes from his design:

TicTacToeRegels
+IsValideZet(Move)
+VerwerkZet(Move m, Board b)

TicTacToeBord
+ grote : int
+ getVakjes()
+ teken()

How could you incorporate this in the rest of the project, without having to change Bertrand's code. Which design pattern would you use?

- (c) (8 points) "The Abstract Factory Pattern embodies the design principle that we should program to an interface, and not an implementation." What does this statement mean? Do you agree or not? Illustrate your answer with references to the design you made in part (a).

Question 6: Closing (3 points)

- (a) (2 points (bonus)) Of the guest lectures we saw, none used the RUP. Instead, most companies used a different software development methodology. Which one?

- (b) (3 points) What is the most important thing you learned in this course?

