CITY UNIVERSITY OF HONG KONG

Course code & title : CS3342 Software Design

Session : Semester B 2013/14

Time allowed : Two hours

This paper has 21 pages (including this cover page).

---

1. This paper consists of 7 questions.
2. Answer ALL questions within the examination booklet.

---

*This is a **closed-book** examination.*

No materials or aids are allowed during the whole examination. If any unauthorized materials or aids are found on a candidate during the examination, the candidate will be subject to disciplinary action.

Student Number: _____

Programme: _____

Seat Number: _____

	Answer all questions							
CILO	*1*	*4*	*4*	*3*	*5*	*3*	*2*	
Question	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**Total**
Max	**10**	**10**	**25**	**10**	**5**	**20**	**20**	**100**

1. Software Development Process (10 Marks)

(a) Give an example to explain the differences between ***prototyping*** and the ***incremental development model*** are. You may illustrate your explanations via diagrams supplemented with texts. **(6 marks)**

(b) *John* is a member of a software development team, he expresses the following:

- The use of the *prototyping approach* is more suitable than the use of the *incremental development model* for their software development needs. He further explains that given all members are junior programmers, and therefore they are unable to produce good requirements documents.
- After a second thought, he also mentions that it is exactly because of their lack of good skills in collecting and documenting the requirements, the team should start with collecting requirements from the users so that the team can easily develop the code that fits the users' needs.

Assuming that you are also a team member of this development team, and want to provide your expert advice to John. Give **one** reason on why the *prototyping approach* is more suitable than the *incremental development model*, and **one** reason on why the *incremental development model* than the *prototyping approach*. Your answers must be related to John's arguments. **(4 marks)**

Reason 1:

Reason 2:

2. Software Design Principles (10 marks)

(a) Sketch a simple software coding example (in either in Java or C++) to illustrate a software design that satisfies the Open-Closed Principle (OCP). **(6 marks)**

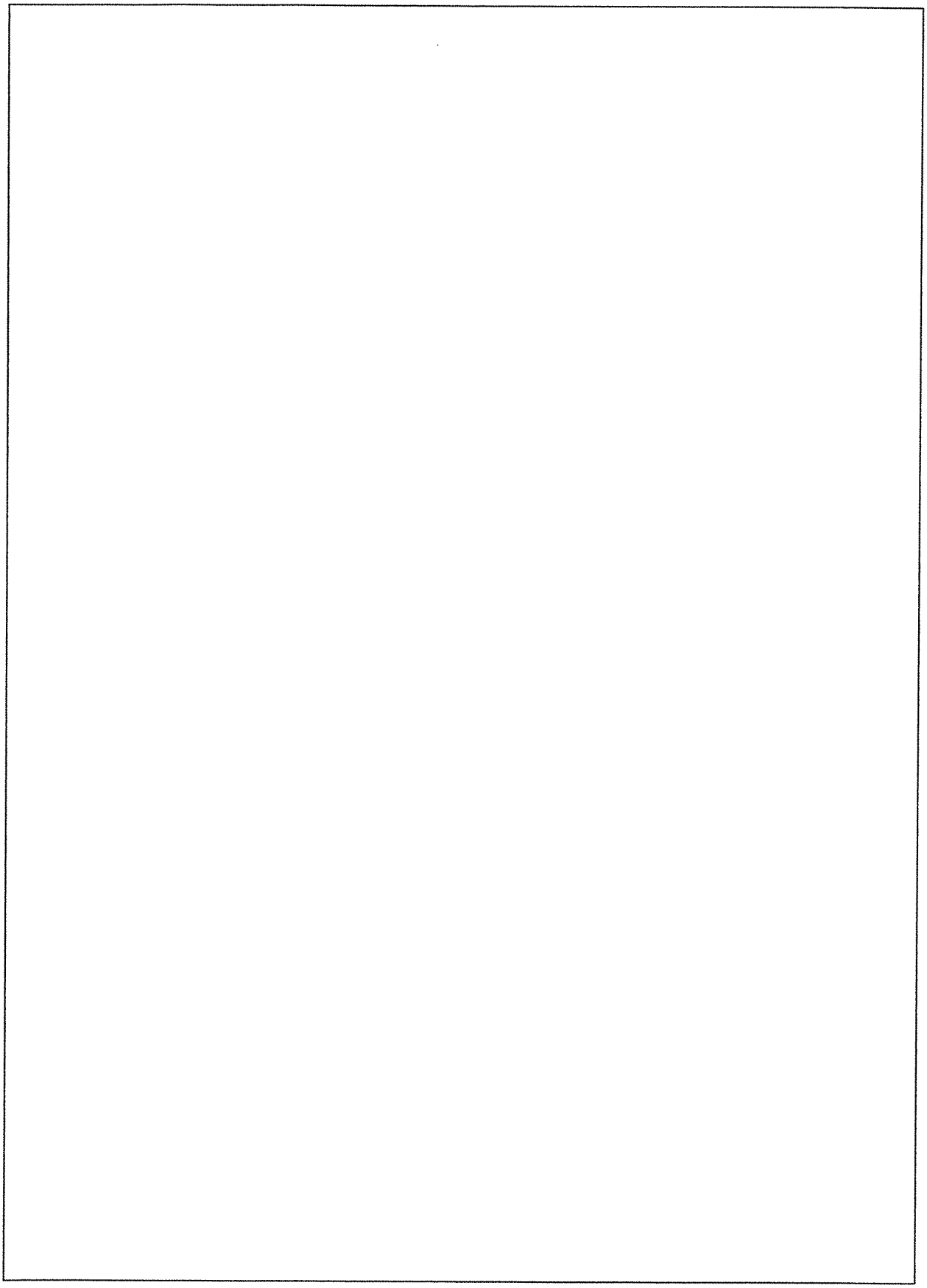
(b) Explain how your coding example in the answer of Question 6(a) has satisfied the Open-Closed Principle (OCP). **(3 marks)**

(c) In software design, can we satisfy Open-Closed Principle (OCP) without using ***interface*** or ***abstract class***? Justify your answer. **(1 mark)**

3. Design Principle and Design Patterns (25 marks)

Study the **Scenario *A*** in Appendix 1, and create a design solution in the form of UML class diagram to solve **ALL** of the following issues on top of the Scenario *A*. Your solution must show all classes, relationships, attributes and methods. Moreover, your solution must be based on good design principles and/or utilize useful design patterns.

- **Issue 1:** All inheritance class hierarchies in the solution should **not violate** both the Liskov Substitution Principle (LSP) and Interface Segregation Principle (ISP).
- **Issue 2:** If a `PaymentRecord` object instance is cancelled, all the associating `SalesRecord` object instances should also be cancelled. However, since all pieces of transaction information are important, no object can be deleted from the system.
- **Issue 3:** The `CashierRegister` class is a higher-level module than that of all other classes. As such, `CashierRegister` should not be able to directly create the object instances of the `SalesRecord` and `PaymentRecord` classes.
- **Issue 4:** Each `CashierRegister` object instance should maintain a list of `PaymentRecord` object instances.



4. Roles of Variables (10 marks)

(a) Appendix 2 shows the definition of each role of variable. **Write a small program** (in either Java or C++) that satisfies the following requirements:

- At least one variable taking the role of Organizer
- At least one variable taking the role of Follower
- No variable takes more than one role.

You must **clearly point out** in your code, which variable refers to as which *role of variable*.

(5 Marks)

(b) Based on the following code fragments, correctly **classify** the role of each variable in the table below. (5 Marks)

Java Version

```
import java.util.*;
class  Average {

  public static void main(String[] args) {
     int MAX = Integer.parseInt(args[0]);
     Scanner scan_input=new Scanner(System.in);
     double input=0,sum=0,average=0;

     System.out.printf("Enter %d numbers:\n",MAX);
     for (int i=0; i<MAX; i++) {
        input = scan_input.nextInt();
        sum += input;
     }
     average = sum / MAX;
     System.out.printf("Average Number: is:
                              %.2f\n",average);
   }
}
```

C++ Version

```
#include <iostream>
using namespace std;

int main(int argc, char* argv[]) {
  int MAX = atoi(argv[1]);

  double input=0, sum=0, average=0;

  printf ("Enter %d numbers:\n", MAX);
  for (int i=0; i<MAX; i++) {
      cin >> input;
      sum += input;
  }
  average = sum / MAX;
  printf ("Average number is: %.2f\n", average);
  return 0;
}
```

Variable	*Role*
MAX	
input	
i	
sum	
average	

5. Professional Ethics in Software Engineering (5 marks)

(a) Study **Scenario *B*** below. **Explain** at least **two** codes of ethics *Mary* violates; and **explain** at least **three** codes of ethics *Angela* follows. Appendix 3 lists out the eight areas of code of ethics in software engineering.

Scenario *B*: In a software development project, *Mary* is the programmer, and *Angela* supervises *Mary*. They are good friends of one another. *Sam* is the user of the software.

- *Mary* presents a user interface (UI) of the software to *Sam*, but *Sam* considers the UI very bad in design. Because *Mary* does not want to make changes, she tells *Sam* that he has to follow the current way to input data.
- *Mary* reports to *Angela* that *Sam* is happy with the current UI.
- *Angela* is surprised by *Mary*'s report because *Angela* tried the UI herself, but did not consider the current UI easy to use. She thus contacts *Sam* directly and finds out that *Sam* also finds the UI is not usable.
- *Angela* therefore coaches *Mary* about professionalism, and suggests *Mary* to consult *Sam* again and improve the UI accordingly.

***Mary* violates:**

***Angela* follows:**

6. Factoring Design Constraints (20 marks)

Scenario *C*: Consider a web-based CityU Room Booking Application

- A student enters the Student number, room size, the nature of the activity, and the period of booking on the *Room-Enquiry Screen* of the system. This screen then displays a list of rooms, in which each room satisfies the input requirements.
- If the student places the mouse over any room item on this room list, the system will both retrieve the corresponding room details, including photos, a floor plan and the blog messages posted by other users of the same room and display them with a [Book] button on the *Room-Detail Window*.
- If a student presses the [Book] button, the system will create a room booking record on the selected room for this student.

(a) Identify all the classes in **Scenario *C***. For each of your identified classes, identity its **analysis class stereotype** (*Entity class*, *Boundary class*, and *Control class*). State the responsibility of each class according to **Scenario *C***. Show your answer in the following table. Note that the number of rows in the table is independent to the number of classes in your solution. **(10 marks)**

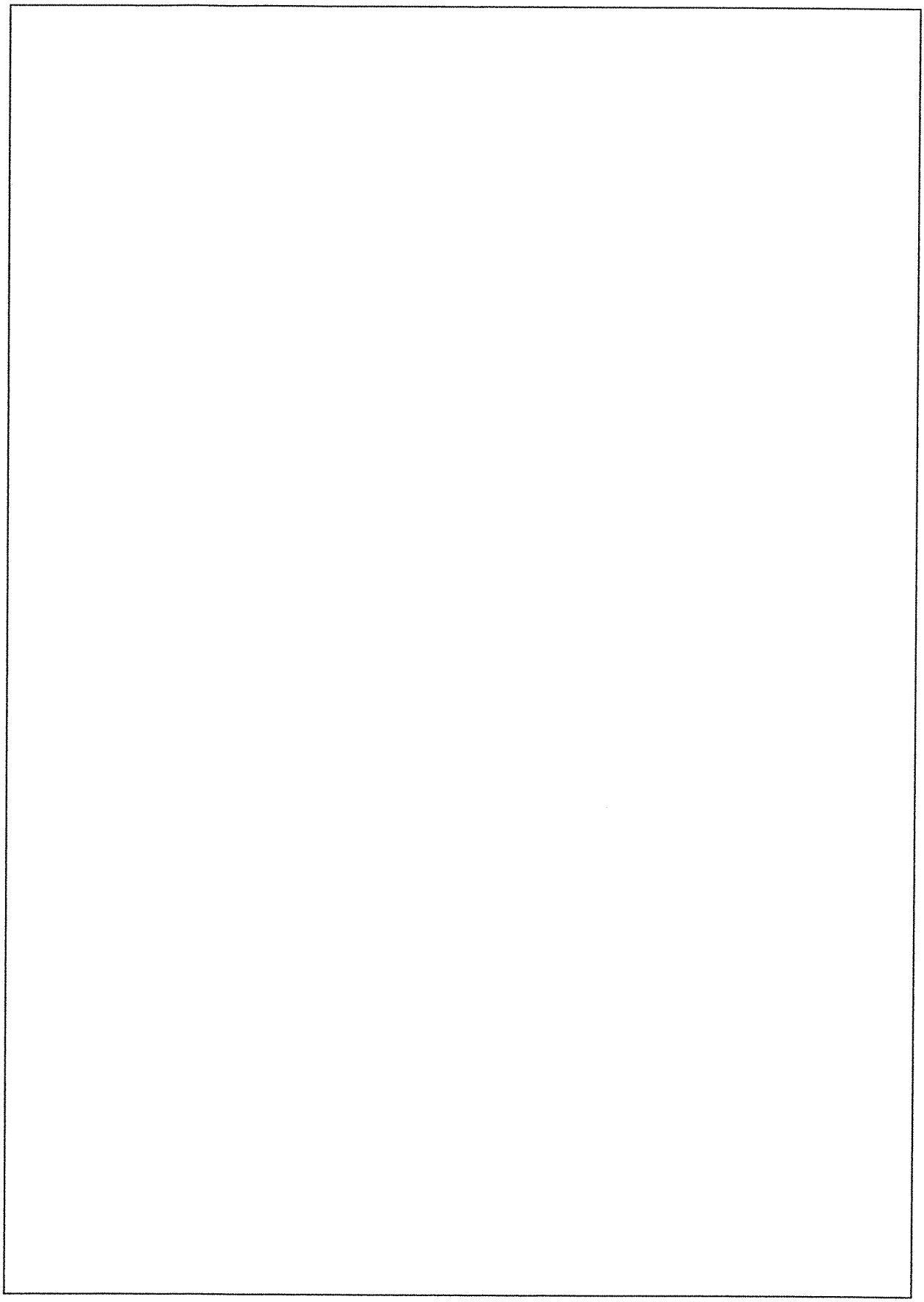
Class	**Analysis Class Stereotype**	**Responsibility of the Class**

(b) Design your solution that factors in the ***Design Constraint 1*** below in ***class diagram*** (using ***analysis class stereotypes***) **OR** ***sequence diagram.*** In addition, **explain** your solution on why it handles this design constraint. (**10 marks**)

Note the following:

- You may add comments to the diagram(s) to clarify your intended solution.

- **Design Constraint 1**: The system should allow displaying multiple *Room-Detail Windows* for a student to compare different rooms. At the same time, if a booking on one room is made, all other *Room-Detail Windows* should disable the corresponding [Book] buttons on them to prevent the student to book multiple rooms for the same period by unintended mistakes.



7. User Requirements Capturing and Specifications (20 marks)

The use case **UC1** in **Appendix 4** illustrates the basic flow for a student to take a computer-based examination in the Course Administration domain at CityU.

(a) Use the **Class-Responsibility-Collaborator (CRC) Card** notations to show the system information described according to the **use case UC1**. [*Note: You may or may not use up all four CRC cards.* If you need more space use the space on page 17.] **(10 marks)**

CRC Card 1:

Class Name:	
Description:	
Responsibilities	Collaborator

CRC Card 2:

Class Name:	
Description:	
Responsibilities	Collaborator

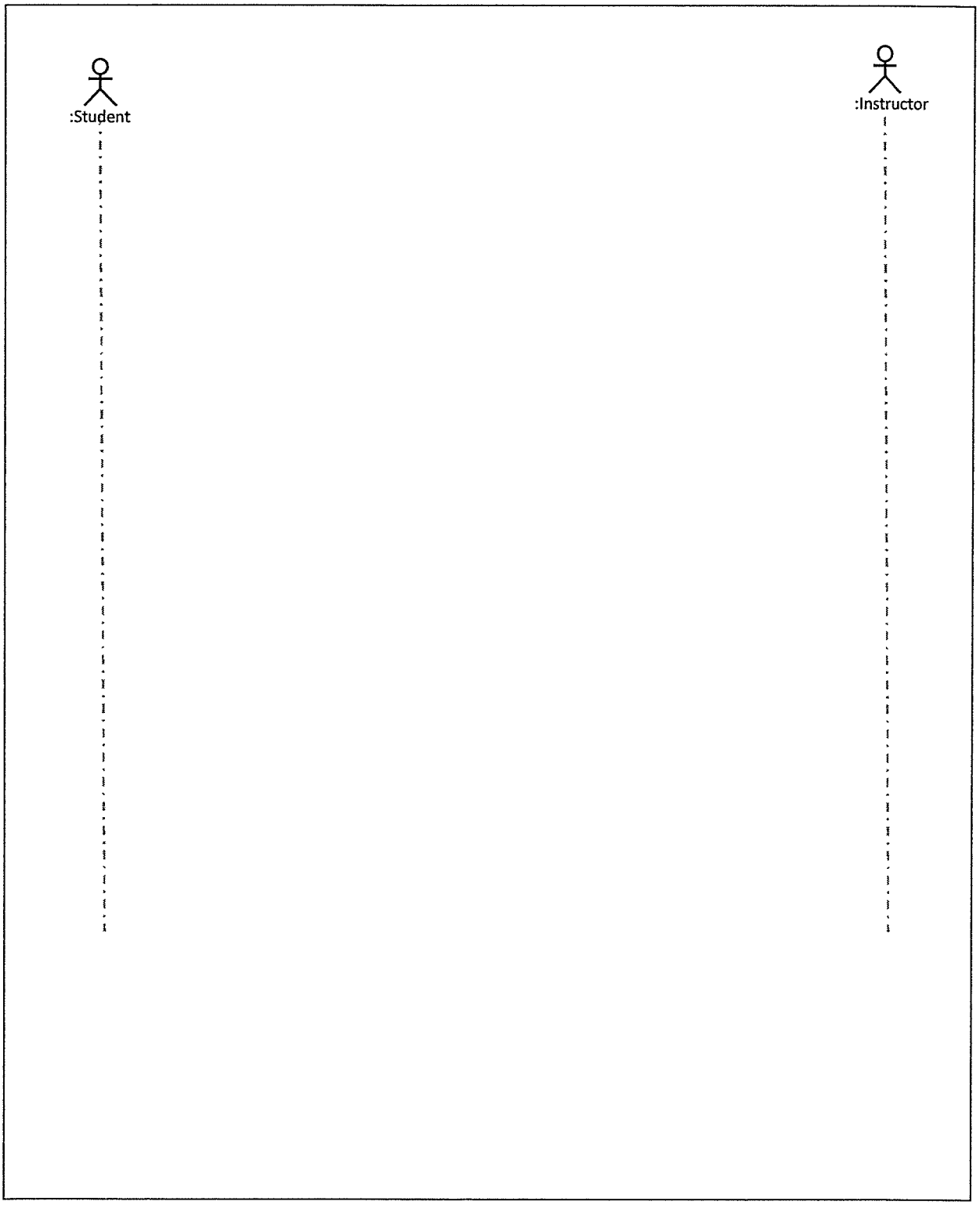
CRC Card 3:

Class Name:	
Description:	
Responsibilities	Collaborator

CRC Card 4:

Class Name:	
Description:	
Responsibilities	Collaborator

(b) Design the object interactions among the classes identified in your CRC cards via a UML **Sequence diagram** so that these objects can fulfill the requirement stated in the use case **UC1** in **Appendix 4**. [Note: Two actors ***Student*** and ***Course Instructor*** have been given to you.] **(10 marks)**



- END -

Note: *This page is intentionally left blank. You may use this page to show additional information that you want to tell the examiner about your answers. It is not a part of any question unless explicitly indicated otherwise.*

Question: _____

Question: _____

Appendix 1: **Scenario *A***

In a supermarket, each cashier operates a Cashier Register, which supports a number of operations:

- The `newSales()` operation creates a blank new `SalesRecord` of the current date.
- The `completeSales()` operation calls the `endRecord()` operation of `Record`.
- The `addSalesItem()` operation calls the `addItem()` operation of `Record`.
- The `makePayment(x, payno)` operation creates a blank new `PaymentRecord` of the current date and with the payment number `pno` followed by calling the `pay(x)` operation of `Record`. In `PaymentRecord`, `pay(int amt)` simply assigns `amt` to its attribute `PaidAmount`. The constructor of `PaymentRecord` assigns the input value to the attribute `PaymentNo`.
- The `cancelPayment()` operation also calls the `endRecord()` operation of `Record`.
- The method bodies of both `pay()` of `SalesRecord` and `addItem()` of `PaymentRecord` contain no any code.

Moreover, multiple `SalesRecord` object instances can be recorded and associate with one `PaymentRecord` object instance. This can be done by performing a few pairs of `newSales()` and `completeSales()` operations followed by one `makePayment(x, payno)` operation.

All classes except `Record` in the scenario are concrete classes.

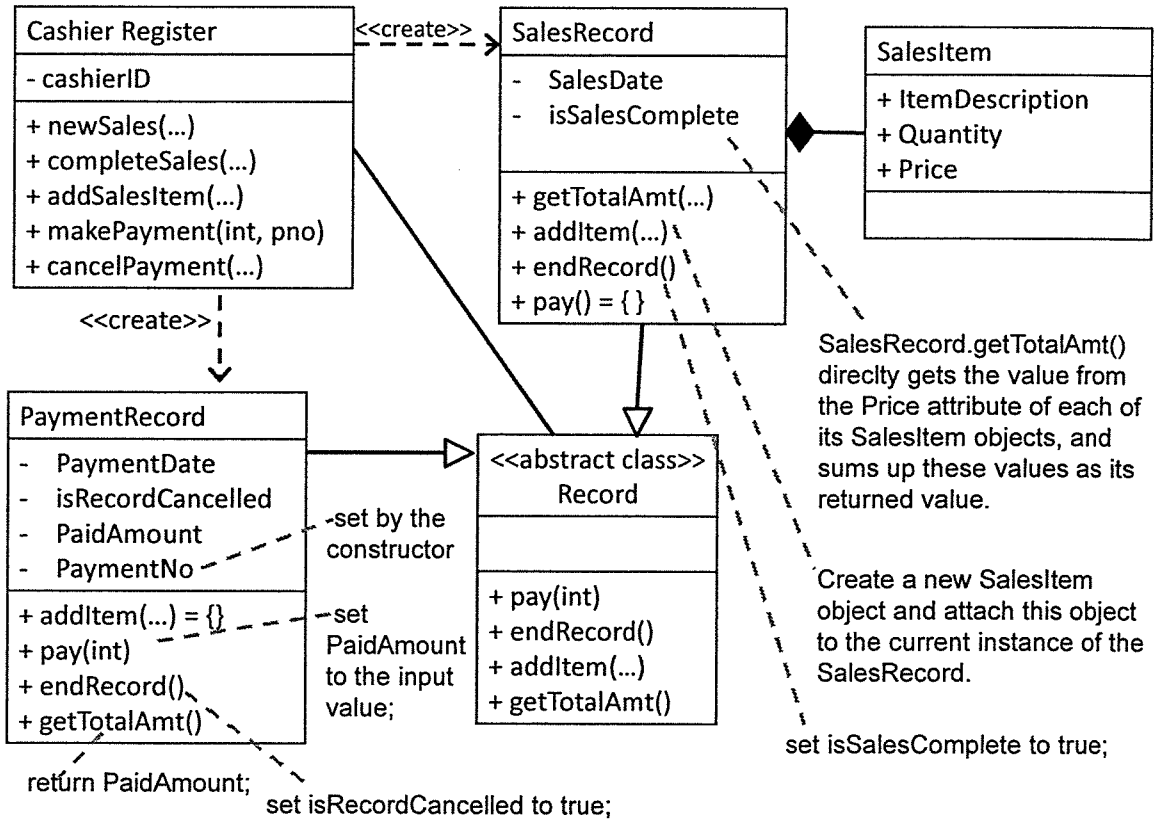


Figure 1. A Class Diagram for Cashier Register Domain

Appendix 2: **Roles for Variables in Software Programs**

Role	**Description**
Constant/ Fixed value	A variable which is initialized without any calculation and whose value does not change thereafter.
Stepper	A variable stepping through values that can be predicted as soon as the succession starts.
Most-recent holder	A variable holding the latest value encountered in going through a succession of values.
Most-wanted holder	A variable holding the "best" value encountered so far in going through a succession of values. There are no restrictions on how to measure the goodness of a value.
Gatherer	A variable accumulating the effect of individual values in going through a succession of values.
Transformation	A variable that always gets its new value from the same calculation from value(s) of other variable(s).
Follower	A variable that gets its values by following another variable.
One-way flag	A two-valued variable that cannot get its initial value once its value has been changed.
Temporary	A variable holding some value for a very short time only.
Organizer	An array which is only used for rearranging its elements after initialization.

Appendix 3: **Code of Ethics in Software Engineering**

Software engineers shall, in their work capacity,

1) [*Public interest*] Act consistently with public interest
2) [*Client and employer*] Act in the best interests of their clients and employer
3) [*Product*] Develop and maintain the product (e.g., software and documentation) with the highest standards possible
4) [*Judgment*] Maintain integrity and independence (of oneself)
5) [*Management*] Promote an ethical (e.g., equal opportunity, match task against skill level instead of friendship) approach in management of subordinates (who are managed by you)
6) [*Profession*] Advance the integrity and reputation of the profession as software engineers
7) [*Colleagues*] Be fair and supportive to colleagues
8) [*Self*] Participate in lifelong learning (as technology changes fast)

Appendix 4: **Use Case (UC1)**

Use Case UC1: Take Computer-based Examination	
Basic Flow	
Actor Action (or Intention)	*System Responsibility*
1. *Student* arrives at an examination venue and signs on the system with the student ID	
	2. Recognize the student's face according to the student ID. 3. Verify the student identity being valid for the examination venue. 4. Confirm that the student has met the coursework requirement of the course. 5. Randomly present an applicable examination paper to the student.
6. *Student* fills in the answers for the examination paper within the allowable time period of the exam	
	7. Record the student answers as a part of the student performance for the course 8. Calculate the mark that the student receives from the filled answer. 9. Map the examination mark to a grade. 10. Notify *Course instructor* about the performance of the student 11. Generate a grade sheet to the student
12. *Student* leaves the examination venue with a grade sheet.	

- END OF APPENDICES-