Exam 11 November 2018, questions and answers

Object Oriented Design (Monash University)

Office Use Only

MONASH University

Semester Two 2018
Examination Period

Faculty of Information Technology

EXAM CODES: **FIT2099**

TITLE OF PAPER: Object Oriented Design and Implementation - PAPER 1

EXAM DURATION: 2 hours writing time

READING TIME: 10 minutes

THIS PAPER IS FOR STUDENTS STUDYING AT: (tick where applicable)

☐ Caulfield ☑ Clayton ☐ Parkville ☐ Peninsula
☐ Monash Extension ☐ Off Campus Learning ☐ Malaysia ☐ Sth Africa
☐ Other (specify)

During an exam, you must not have in your possession any item/material that has not been authorised for your exam. This includes books, notes, paper, electronic device/s, mobile phone, smart watch/device, calculator, pencil case, or writing on any part of your body. Any authorised items are listed below. Items/materials on your desk, chair, in your clothing or otherwise on your person will be deemed to be in your possession.

No examination materials are to be removed from the room. This includes retaining, copying, memorising or noting down content of exam material for personal use or to share with any other person by any means following your exam.
Failure to comply with the above instructions, or attempting to cheat or cheating in an exam is a discipline offence under Part 7 of the Monash University (Council) Regulations, or a breach of instructions under Part 3 of the Monash University (Academic Board) Regulations.

AUTHORISED MATERIALS

OPEN BOOK	☑ **YES**	☐ **NO**
CALCULATORS	☐ **YES**	☑ **NO**
SPECIFICALLY PERMITTED ITEMS	☐ **YES**	☑ **NO**

if yes, items permitted are:

Candidates must complete this section if required to write answers within this paper

STUDENT ID: __ __ __ __ __ __ __ __ DESK NUMBER: __ __ __ __ __

Question 1. **[Total Marks 4 + 2 + 2 + 2 = 10]**

Consider the following code for an Java class `Unit` from Java system that represents information about Units, Students, and Marks from a University with structures like those of Monash (continued on following pages):

```
import java.util.ArrayList;
import java.util.HashMap;

public class Unit {

	private String code;
	private String name;
	private HashMap<Integer, Student> enrolledStudents = new HashMap<Integer, Student>();
	private AssessmentScheme assessmentScheme = null;
	private HashMap<Assessment, HashMap<Student, Mark> > Marks
		= new HashMap<Assessment, HashMap<Student, Mark> >();

	public Unit(String newCode, String newName) {
		code = newCode;
		name = newName;
	}

	public void enrolStudent(Student newStudent) {
		enrolledStudents.put(newStudent.getPersonID(), newStudent);
	}

	public void unenrolStudent(Student student) {
		enrolledStudents.remove(student.getPersonID());
	}

	public boolean isEnrolled(Student student) {
		return enrolledStudents.containsKey(student.getPersonID());
	}

	public ArrayList<Student> getEnrolledStudents() {
		ArrayList<Student> students = new ArrayList<Student>(enrolledStudents.values());
		return students;
	}

	public void setAssessmentScheme(AssessmentScheme assessmentScheme) {

		this.assessmentScheme = new AssessmentScheme(assessmentScheme);
		for (Assessment a : assessmentScheme.getAssessments()) {
			Marks.put(a, new HashMap<Student, Mark>());
		}
	}

	public AssessmentScheme getAssessmentScheme() {
		return new AssessmentScheme(assessmentScheme);
	}

	public boolean hasAssessmentScheme() {
		return assessmentScheme != null;
	}


	public boolean hasCompletedAssessments(Student student) {
		boolean hasCompleted = true;
		for (Assessment a : assessmentScheme.getAssessments()) {
			hasCompleted &= Marks.get(a).containsKey(student);
		}
		return hasCompleted;
	}
```

```
	public void markStudent(Assessment assessment, Student student, int score, String
comment) throws Exception {
		/* Start Preconditions */
		// Precondition: studentEnrolledInUnit
		if (! isEnrolled(student)) {
			throw new Exception("Precondition violated: studentEnrolledInUnit");
		}
		// Precondition: scoreInValidRange
		if (
			(score < 0) // too low
			 || (score > assessment.getWeight())  // too high
		   ) {
			throw new Exception("Precondition violated: scoreInValidRange");
		}
		/* End Preconditions */

		Mark mark = new Mark(assessment, student, score, comment);
		Marks.get(assessment).put(student, mark);
	}

	public HashMap<Student, Mark> getMarks(Assessment assessment) {
		return (HashMap<Student, Mark>) Marks.get(assessment).clone();
	}

	public int getUnitMark(Student student) throws Exception {
		/* Start Preconditions  */
		// Precondition: hasCompleted
		if (!hasCompletedAssessments(student)) {
			throw new Exception("Precondition violated: hasCompleted");
		}
		/* End Preconditions */

		int unitMark = 0;
		for (Assessment a : assessmentScheme.getAssessments()) {
			unitMark += Marks.get(a).get(student).getScore();
		}
		return unitMark;
	}

	public String description() {

		return code + " " + name;
	}

}
```

Question 1. (continued)

(a) Why is it considered good practice to keep the implementation details of classes private? Give two examples of problems that could arise during software development and maintenance if implementation details were not kept private. Illustrate your answer with examples from the `Unit` class above. **(4 marks)**

(b) Why does the method

```
getEnrolledStudents()
```

return a newly-created `ArrayList`? What would be the risks if this method returned a reference to the existing `enrolledStudents` attribute, e.g.

```
return this.enrolledStudents;
```
? **(2 marks)**

(c) The precondition code in methods `markStudent(…)` and `getUnitMark(…)` makes use of other method of the class: `isEnrolled(…)` and `hasCompletedAssessments(…)`.

Should methods used in precondition code, such as `isEnrolled(…)` and `hasCompletedAssessments(…)`, be public or private? Explain the reasons for your answer.

(2 marks)

(d) If you find duplicated code in an object-oriented program, there are two basic ways of reorganizing the code so that the required code appears in one place only. Explain these two basic ways of eliminating duplicated code. **(2 marks)**

Question 2. **[Total Marks 3 + 1 + 2 + 2 + 2 = 10]**

Design By Contract is an approach to software design that is supported directly in some languages, and which can be applied in others, such as Java, through the use of extensions or other language features. In Design By Contract, the designer of a class tells the clients of the class what the class does and what it needs via the *specification* of the class.

An important principle in Design By Contract is that of *Command-Query Separation*.

(a) What is Command-Query Separation? Why is it useful in Design By Contract? Give an example of a situation where not using Command-Query Separation could cause a problem. **(3 marks)**

Another design principle that is used in Object-Oriented design is the *Liskov Substitution Principle*.

(b) What is the Liskov Substitution Principle? **(1 marks)**

(c) What is the impact of the Liskov Substitution Principle on the rules relating pre- and post-conditions of superclass methods and subclass methods in Design By Contract? **(2 marks)**

(d) Why should a method not to try to cope with a violation of its preconditions, but simply to report the problem to the client that called it? What problems could arise if it did not do this? **(2 marks)**

(e) Why might it be useful to write the preconditions for a method before it is implemented? Why might this be easier than simply writing the method? Illustrate your answer with an example **(2 marks)**

Question 3. [Total Marks 2 + 3 + 5 = 10]

In you assignments, you worked on a rogue-like game set in the Harry Potter universe. Imagine now that you're working on another rogue-like game, this time set in a world of superheros like Spider-man or Batman. In this game, some, but not all, characters can have one or more superpower. Every superpower is implemented by a different class in the system. Superpowers include becoming invisible, shooting webs, jumping tall buildings, and so on.

To implement this functionality, you decide that those characters with superpowers need to have some kind of standard way to choose and execute one of their superpowers; that is:

- Choose a superpower
- Set up and execute the superpower
    - The requirements for setting up and executing difference superpowers differ. For example, to set up shooting a web, it is necessary first to choose a target. Becoming invisible, however, does not require this.

You are considering the best design for giving characters these abilities. Two alternatives occur to you:

- Creating an interface that all characters with superpowers must `implement`.
- Creating an abstract class that all characters with superpowers must `extend`.

(a) What is an *interface* in Java? Illustrate your answer with a code example for the scenario above. **(2 marks)**

(b) What is an *abstract class* in Java? How does an abstract class differ from an interface? Illustrate your answer with a code example for the scenario above. **(3 marks)**

(c) Explain the advantages and disadvantages of using interfaces and abstract classes to implement the superpower functionality for characters described above. Which would choose, and why? **(5 marks)**

Question 4. **[Total Marks: 2 + 2 + 2 + 4 = 10 marks]**

A key principle in software engineering is that we should reduce dependencies between software elements whenever possible.

(a) Explain why is it important to reduce dependencies between software elements whenever possible. Include an example of a real-world problem that could occur if this is not done. **(2 marks)**

(b) Explain how encapsulation and language features such as access modifiers can be used to reduce and control dependencies. Provide a Java example that illustrates your explanation. **(2 marks)**

(c) Explain what it means to find a good abstraction for use in a program. How does this differ from the notion of abstract classes? **(2 marks)**

Modern computers can use many different kinds of storage devices (e.g. flash drives, spinning hard drives, solid state disks).

(d) Explain how the Dependency Inversion Principle makes it possible for us to write code to store data to these devices without needing to know what type of physical device we are actually using. Include a UML class diagram to illustrate your answer. **(4 marks)**

Question 5. **[Total Marks: 5 + 5 + 5 + 5 = 20 marks]**

Consider the code below and on the following pages from an information system for a library that currently lends out books and magazines.

```
public class Library {

	private static int nextID = 1;
	public static final int bookLoanLength = 14; // max loan length for books in days
	public static final int magazineLoanLength = 7; // max loan length for magazines
in days

	public Library() {
		// nothing to do in constructor at this stage
	}

	public static int newID() {
		return(nextID++);
	}

	public static final int bookLateCharge(int daysLate) {
		// charge per day late in cents
		return 100*daysLate;
	}

	public static final int magazineLateCharge(int daysLate) {
		// charge per day late in cents
		return 50*daysLate;
	}
}
```

```
public class Book {

	private int id; // unique id of product in library
	private String title; // short name to appear in catalogue
	private String author;
	private String summary;

	public Book(String title, String author, String summary) {
		this.title = title;
		this.author = author;
		this.summary = summary;
		this.id = Library.newID();
	}

	public int getId() {
		return id;
	}

	public String getTitle() {
		return title;
	}

	public String description() {
		String description = id + ": " + title + "\n"
			+ "by " + author + "\n"
			+ summary;
		return description;
	}

}
```

```
public class Magazine {

	private int id; // unique id of product in library
	private String title; // short name to appear in catalogue
	private int volume;
	private int number;

	public Magazine(String title, int volume, int number) {
		this.title = title;
		this.volume = volume;
		this.number = number;
		this.id = Library.newID();
	}

	public int getId() {
		return id;
	}

	public String getTitle() {
		return title;
	}

	public String description() {
		String description = id + ": " + title
			+ " (vol. " + volume
			+ ", num. " + number + ")";
		return description;
	}

}
```

```
import java.util.Date;
import java.util.HashMap;
import java.util.concurrent.TimeUnit;

public class LoanRecord {

	private HashMap<Book, Date> booksOnLoan = new HashMap<Book, Date>();
	private HashMap<Magazine, Date> magazinesOnLoan = new HashMap<Magazine, Date>();


	public void addBook(Book bookOnLoan) {
		Date now = new java.util.Date();
		booksOnLoan.put(bookOnLoan, now);
	}

	public void addMagazine(Magazine compactDiscOnLoan) {
		Date now = new java.util.Date();
		magazinesOnLoan.put(compactDiscOnLoan, now);
	}

	/* Useful for testing */
	public void addBookWithStartDate(Book bookOnLoan, Date startDate) {
		booksOnLoan.put(bookOnLoan, startDate);
	}
```

```
	public int overdueCharge() {
		int totalCharge = 0;
		for (Book book : booksOnLoan.keySet()) {
			int daysOverdue =  this.daysOnLoanBook(book) - Library.bookLoanLength;
			if (daysOverdue > 0) {
				totalCharge += Library.bookLateCharge(daysOverdue);
			}
		}
		for (Magazine magazine : magazinesOnLoan.keySet()) {
			int daysOverdue =  this.daysOnLoanMagazine(magazine) -
Library.magazineLoanLength;
			if (daysOverdue > 0) {
				totalCharge += Library.magazineLateCharge(daysOverdue);
			}
		}
		return totalCharge;
	}

	public String contentsDescription() {
		String description = "";
		for (Book book : booksOnLoan.keySet()) {
			description += String.format("%s\t%d days on loan\n",
					book.getTitle(),
					this.daysOnLoanBook(book)
			);
		}
		for (Magazine magazine : magazinesOnLoan.keySet()) {
			description += String.format("%s\t%d days on loan\n",
					magazine.getTitle(),
					this.daysOnLoanMagazine(magazine)
			);
		}
		return description;
	}

	private int daysOnLoanBook(Book book) {
		Date now = new java.util.Date();
		// calculate different between loan time and now in milliseconds
		long timeOnLoan = now.getTime() - booksOnLoan.get(book).getTime();
		// convert to days
		long timeOnLoanInDays = TimeUnit.DAYS.convert(timeOnLoan,
TimeUnit.MILLISECONDS);
		return (int)timeOnLoanInDays;
	}

	private int daysOnLoanMagazine(Magazine magazine) {
		Date now = new java.util.Date();
		// calculate different between loan time and now in milliseconds
		long timeOnLoan = now.getTime() - magazinesOnLoan.get(magazine).getTime();
		// convert to days
		long timeOnLoanInDays = TimeUnit.DAYS.convert(timeOnLoan,
TimeUnit.MILLISECONDS);
		return (int)timeOnLoanInDays;
	}

}
```

```java
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;

public class TestLibrary {

	public static void main(String[] args) throws ParseException {

		// Create some library items
		Book cloudstreet = new Book(
			"Cloudstreet", // title
			"Tim Winton", // author
			"Winner of the Miles Franklin Award..." // summary
		);

		Magazine ieeeSoftware1_2 = new Magazine(
			"IEEE Software", // title
			1, // volume
			2 // number
		);

		// add some items to a loan record
		LoanRecord loanRecord = new LoanRecord();
		// add a book with loan starting on 1st of September 2018
		SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy", Locale.ENGLISH);
		Date startDate = sdf.parse("01/09/2018");
		loanRecord.addBookWithStartDate(cloudstreet, startDate);
		// add a magazine with loan starting now
		loanRecord.addMagazine(ieeeSoftware1_2);

		// show description of loan record, and overdue charges
		System.out.println("Loan Description:\n" + loanRecord.contentsDescription());
		float overdueCharge = loanRecord.overdueCharge()/100;
		System.out.println("Overdue Charges: $"
			+ String.format("%.2f", overdueCharge) + "\n");
	}

}
```

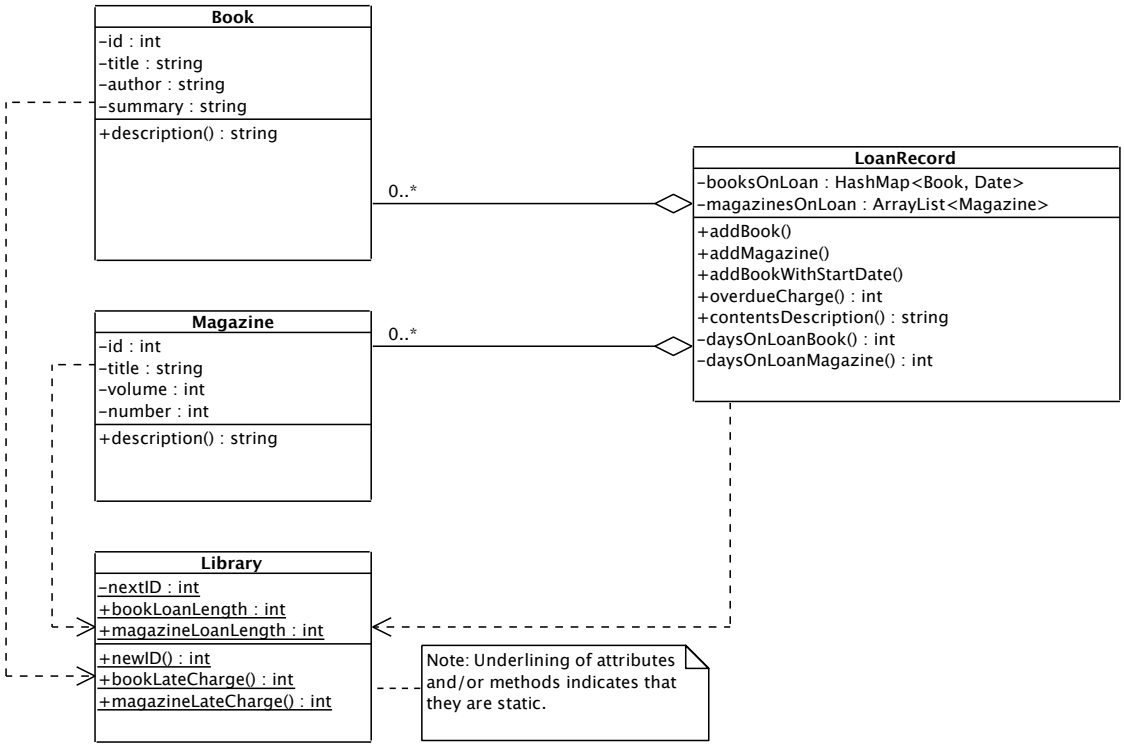
Question 5. (continued)

(a) What code and design smells do you detect in classes `Library` and `LoanRecord`? Explain the problems you see, and why they are bad. **(5 marks)**

(b) Imagine that the library now decides that it is going to start lending DVDs. DVDs have an ID number, a title, a director, and a star. Explain why supporting DVDs would be painful with the current design of the class `LoanRecord`. **(5 marks)**

Question 5. (continued)

The UML class diagram below shows the current design of this part of the system (getter and setter methods are not shown, but can be assumed to exist). Your task is to refactor this design to solve the problems you identified in parts (a) and (b).



(c) Draw a UML class diagram showing your proposed redesign of this part of the system to address the problems you identified in parts (a) and (b) (include a DVD class). Include attributes and methods as above. **(5 marks)**

(d) Write a rationale for the design you showed in part (c). Explain how it addresses the code and design smells you identified earlier, and facilitates the extension of the system to support the addition of DVDs to the library's collection. Also explain any other advantages or disadvantages your proposed design. **(5 marks)**