# CS2102, B11

# Exam 2

---

Name:

---

You have 50 minutes to complete the problems on the following pages. There should be sufficient space provided for your answers.

If a problem asks you to create an interface, you should provide a complete interface, including method headers and argument types.

If a problem asks you to create a class:

- Include **implements** and **extends** statements
- Include field names and types
- Omit constructors
- Omit methods unless a problem asks otherwise

Omit the `Examples` class (examples of data and test cases) unless a question asks otherwise.

---

1. **Topic: Access Modifiers and Encapsulation**

   You are implementing a gradebook for a course. The gradebook stores each student's grades and has a method for producing course grades for the university. The current set of Java classes (given below) assumes that a student's grades will only consist of two exam grades. Ideally, the gradebook should allow different courses to use different grading systems, while still producing the same kind of grade reports for the university.

```
// a class for reporting course grades to the university
// grades are strings such as "A", "B", "pass", "NR"
class CourseGrade {
  String name;
  String grade;
}

// a class for storing one student's performance in the course
class Student {
  String name;
  int midtermGrade;  // grade on the midterm
  int finalGrade;    // grade on the final
}

// the course gradebook
class Gradebook {
  _________ LinkedList<Student> students;

  // produces list of CourseGrades, one per student in the course
  _________ LinkedList<CourseGrade> computeCourseGrades() {
    LinkedList<CourseGrade> grades = new LinkedList<CourseGrade>();
    for (Student s : this.students) {
      grades.add(new CourseGrade(s.name, this.computeGrade(s)));
    }
    return grades;
  }

  // computes either "pass" or "NR" as grade for the given student
  _________ String computeGrade(Student s) {
    int average = Math.round((s.midtermGrade+s.finalGrade)/2);
    if (average >= 55)
      return "Pass";
    else return "NR" ;
  }
}
```

   (a) For the code as written, fill in each blanks in the `Gradebook` class with the appropriate access modifier (public/private/protected) for each field/method. You do not need to write anything else for this part.

   (b) On the code itself, box off the code fragments that need to change in order to encapsulate the grading data stored in the `Student` class. You only need to draw boxes to answer this part.

**(exam continues next page)**

(c) Indicate which classes and/or interfaces you would add in order to encapsulate grades. Provide complete interfaces. For classes, provide headers for methods that are not required by their interface. Include only fields and method headers needed to encapsulate the code on the previous page (don't add new features).

(d) Would your encapsulated code make any changes to the `computeGrade` method? Either explain the changes briefly in text or mark up the original code. Justify your decision briefly in text.

**(exam continues next page)**

2. **Topic: Data Structures and Methods as Arguments**

A web-based email provider allows advertisers to contact users based on attributes of the user, such as their age or the words that appear frequently in their email messages. The attributes and mailboxes of each user are stored in a `User` class:

```
class User {
  String username;
  Date birthdate;
  Mailbox email;
  Words frequentWords;

  void newEmail(String message) { ... }  // put message in mailbox
  int getAge() { ... } // compute age from birthdate
}
```

(a) The `Mailbox` class contains all of a user's email messages. Which of the following data structures would you use to store the messages? Circle **one** choice and briefly justify your answer in text. For facts points on one data structure you did not choose, explain why you ruled out that choice.

List Graph Hashmap Heap Binary Tree

(b) The `Words` class maintains information on how many times various words have appeared in a user's email messages. Which of the following data structures would you use to store word-frequency information? Circle **one** choice and briefly justify your answer in text. For facts points on one data structure you did not choose, explain why you ruled out that choice.

List Graph Hashmap Heap Binary Tree

**(exam continues next page)**

(c) Rather than give advertisers their user database, the email provider sends messages on the advertisers' behalf. Advertisers provide two methods: one to check whether to send the ad to a particular user, and another that generates the text of the ad. These methods are passed as arguments to a `targetUsers` method within the email system:

```
// sends an ad to each user that the ad wants to target
// (assume users is the list of all email users)
void targetUsers(_________ withAd) {
  for (User u : users) {
    if (withAd.shouldSendTo(u))
      u.sendMail(withAd.AdText(u));
  }
}
```

Here is a sample ad (there could be many others):

```
class AdvertiseToysToKids {
  boolean shouldSendTo(User u) {
    return (u.getAge() < 17);
  }

  String AdText(User u) {
    // builds message string with user's name
    return "Hey " + u.username + ", great toys for sale!"
  }
}
```

i. Fill in the type (blank line) for `withAd` in `targetUsers`. If your answer involves a new type, provide the code that defines the new type and mark any edits to the existing code as needed to use the new type.

ii. Provide an expression using `targetUsers` that shows how to send the `AdvertiseToysToKids` ad to appropriate users in the email system.

**(exam continues next page)**

3. **Topic: Graphs, Exceptions, Interfaces, Abstract Classes, and Generics**

An adventure game program has the following class definition for rooms. Each room can have doors to multiple other rooms, but a room can have only one staircase to another room.

```
class Room {
  int money;
  LinkedList<Room> byDoor;
  Room byStairs;

  Room(int amount) {
    this.money = amount;
    this.byDoor = new LinkedList<Room>();
    this.byStairs = null;
  }

  // add a door between this room and another room
  void connectDoor(Room to) {
    this.byDoor.add(to);
    to.byDoor.add(this);
  }

  // connect the stairs between this room and another room
  void connectStairs(Room to) {
    this.byStairs = to;
    to.byStairs = this;
  }
}
```

(a) Why does the `Room` constructor take only the initial amount of money in the room, but not the rooms accessible by doors and stairs?

(b) Since a room can have only one staircase, the program should report a problem if someone tries to connect the stairs on a room whose stairs are already connected. Edit the `connectStairs` method to alert the method that called it when the stairs are already connected to another room. If your approach requires new classes or interfaces, define them as part of your answer. **You do NOT need to write a method that calls `connectStairs` as part of your answer.**

**(exam continues next page)**

(c) After the first version of the game does well, the designers decide to change the `money` field to a more general `prize` field with different kinds of prizes (such as magic spells, swords, extra food, etc). Here are three proposed definitions for the new `prize` field, one using an interface, one using an abstract class, and one using generics:

```
class Room {
  IPrize prize;
  ...
}
```

```
class Room {
  AbsPrize prize;
  ...
}
```

```
class Room<P> {
  P prize;
  ...
}
```

Indicate which one of these three options you would prefer by circling one of their terms below. In the space under each option, provide a sentence explaining either why you chose it or why you ruled it out.

Interface

Abstract Class

Generic

**(end of exam)**

# Grading Summary

| Topic | Max Points | Score |
|---|---|---|
| Q1: What is encapsulation | 5 | |
| Q1: Know how/when to encapsulate code | 20 | |
| Q1: Protect data from access/modification | 5 | |
| Q2: What is a | 5 | |
| Q2: What is a | 5 | |
| Q2: What is a | 5 | |
| Q2: What is a | 5 | |
| Q2: Choose an appropriate data structure | 10 | |
| Q2: Setup for Visitors | 10 | |
| Q2: Pass method as an argument | 3 | |
| Q3: Examples of graphs | 7 | |
| Q3: Know when to throw exceptions | 5 | |
| Q3: Create, throw, catch exceptions | 5 | |
| Q3: When to use interface versus abstract class | 5 | |
| Q3: When to use a generic | 5 | |
| | Total (out of 100) | |

**(This page is not part of the exam)**