Exam 2

Graded

Student

AKSHAJ KAMMARI

Total Points

41.5 / 70 pts

Question 1

UML

9 / 22 pts

+ 22 pts Correct

Book Class

✔ + 0.5 pts Book

+ 0.5 pts Some unique id

✔ + 0.5 pts title

✔ + 0.5 pts author

+ 0.5 pts flag for on shelf/available

Library Class

✔ + 0.5 pts Library

+ 0.5 pts name

✔ + 0.5 pts location/address

Member Class

✔ + 0.5 pts Member

✔ + 0.5 pts name

+ 0.5 pts address

Member registered with home Library association

✔ + 1 pt Simple association

✔ + 0.5 pts `*` on Member side

+ 0.5 pts 1 on Library side

Library owns Book association

+ 1 pt Association

✔ + 2 pts Composition

+ 0.5 pts `1..*` on Book side

Members holds Book association

✔ + 1 pt Simple association

✔ **+ 1 pt** 1..* on Member side

+ 0.5 pts 1..* on Book side

Members borrows Book association

+ 1 pt Simple association

+ 1 pt 1 on Member side

+ 0.5 pts 1..* on Book side

Member-holds-Book association class

+ 2 pts Association class, e.g. Hold

+ 0.5 pts hold date

+ 0.5 pts pickup library

Member-borrows-Book association class

+ 2 pts Association class e.g. Borrow or Loan

+ 0.5 pts borrow date

+ 0.5 pts return y date

+ 0 pts Incorrect

💬 Sanchay Kanade

Question 2

Inheritance

9 / 12 pts

+ 12 pts Correct

Square Class

- ✔ **+ 1 pt** side
- ✔ **+ 1 pt** constructor implementation
- ✔ **+ 1 pt** perimeter or area implementation

Cube extends Square

- ✔ **+ 1 pt** extends
- ✔ **+ 1 pt** constructor calls super(side)

+ 3 pts overrides perimeter or area correctly by using all of superclass implementation plus more

- ✔ **+ 4 pts** Provides additional functionality with new method (e.g. volume)

+ 0 pts Incorrect

Question 3

Object Design

4.5 / 8 pts

+ 8 pts Correct

Employee class

✔ **+ 0.5 pts** name

✔ **+ 2 pts** Constructor with name

✔ **+ 0.5 pts** name getter

Concierge or Receptionist Class

+ 2 pts Employee field

✔ **+ 0.5 pts** Some distinguishing field (e.g. hourlyRate for concierge, or salary for Receptionist)

+ 1.5 pts Constructor with Employee as parameter, initializes employee field

✔ **+ 1 pt** name getter implementation delegates to Employee getter

---

+ 0 pts Incorrect

Question 4

Lambda Expressions **7** / 9 pts

4.1 **(no title)** **0** / 2 pts

+ **2 pts** Correct

+ **0.5 pts** Valid

+ **1.5 pts** `void stuff();`

✔ + **0 pts** Incorrect

4.2 **(no title)** **3** / 3 pts

+ **3 pts** Correct

✔ + **0.5 pts** Valid

✔ + **2.5 pts** `String stuff(boolean i, boolean j);`

+ **0 pts** Incorrect

4.3 **(no title)** **2** / 2 pts

✔ + **0.5 pts** Valid

✔ + **1.5 pts** `int stuff(String s)`

+ **0 pts** Incorrect Answer.

4.4 **(no title)** **2** / 2 pts

✔ + **0.5 pts** Invalid

✔ + **1.5 pts** Either `i -> {return i*2;}` or `i -> i*2`

+ **0 pts** Incorrect Answer.

Question 5

Lambda Expressions **12** / 19 pts

5.1 **(no title)** **4** / 4 pts

✔ **+ 4 pts** `Function<Song,String> genre = s -> s.getGenre();`

+ 3 pts Correct, except Song and String are switched

+ 1 pt Correct RHS, incorrect or missing LHS

+ 0 pts Incorrrect

5.2 **(no title)** **2** / 4 pts

+ 4 pts `BiFunction<String,String,Song> newSong = Song::new`

+ 3 pts Correct, except Song is not last type param

+ 1 pt Correct RHS, missing or incorrect LHS

✔ **+ 0 pts** Incorrect

+ 2 pts correct lhs with last param not Song.

5.3 **(no title)** **6** / 6 pts

+ 6 pts Correct

✔ **+ 1 pt** Predicate for pop song

✔ **+ 1 pt** predicate for rock song

✔ **+ 1 pt** predicate for 10,000 or more copies sold

✔ **+ 3 pts** Composition of predicates

+ 0 pts Incorrect

5.4 **(no title)** **0** / 5 pts

+ 5 pts Correct

+ 2 pts Correct LHS, `Comparator<Song>`

+ 3 pts Correct RHS, either `(s1,s2) -> s1.copiesSold() - s1.copiesSold()` or `Comparator.comparing(Song::copiesSold)` or `Comparator.comparing(s -> s.copiesSold())`

✔ **+ 0 pts** Incorrect

CS 213 Spring '24 : Midterm Exam 2

This exam is worth 70 points. At the end of the term for grade determination, your score here will be doubled to be out of 140 points, for 14% of the course grade.

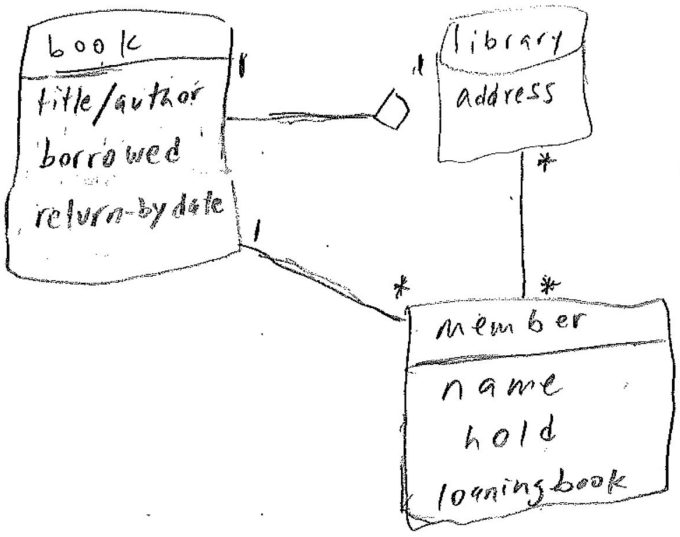
Name: Akshaj Kamman NetID: ak1990

1. **UML (22 pts)**

   Draw the *most appropriate* UML class diagram for a part of an online library system described here. The system is a group of libraries which exist only to lend out books, and each library in the group has at least one book. A member is registered with a home library (the township where they live). Each book has only one copy, and it is owned by one library.

   A member can search for a book (that could be owned by any library) by title/author and put a *hold* on it, and ask to pick it up at any of the libraries. If the book is on the shelves, it is shipped to the pick up library. If it is not available, the member will have to wait for an undetermined number of time until it is available (first come first serve), at which point it is shipped to the pick up library. When the book is available at the pick up library, the member can pick it up, at which time the book is is considered to be borrowed or loaned out and given a return-by date.

   For any association, make sure to show multiplicities. Inside each class, only list the class name and minimal number of fields (data type not needed) required to characterize objects of that class and identify instances uniquely. No access levels or methods needed.



2. **Inheritance (12 pts)**

Although saying "Cube IS A Square" doesn't make sense, it can be implemented in Java using inheritance in a legitimate way. Implement `Square` and `Cube` classes with minimal number of field(s), one constructor, and minimal number of methods (*do NOT implement getters and setters*) required to demonstrate all aspects of legitimate inheritance.

```
Public Class Square () {

int Square (int length) {
    this.length = length;
}

int area (int length) {
return length * length;
}
```

```
Public Class Cube extends Square {

int Cube (int side) {
    super (side)
}

int volume (int edge) {
    return area () * edge;
}
```

3. **Object Design (8 pts)**

Consider a hotel where an employee could be both receptionist (full-time) and concierge (hourly) at the same time, or at different times. Write classes for employee, and *either* receptionist *or* concierge. In each class, write at least one field that is specific to that class, and one constructor. Also, in both classes, write a getter method to return the name.

NetID: ak1990

```
Public class Employee () {
  String name;
  int wage;
  public Employee (String name, int wage) {
  this.name = name;
  this.wage = wage;
  }

public int getName (int name) {
  return name;
  }

Public class receptionist () {
  string name;
  int salary;
  Public Receptionist (String name, int salary) {
    this.name = name;
    this.salary = salary;
    }
  public int getName (int name) { return name; }
```

4. **Lambda Expressions (9 pts, 2+3+2+2)**

For each of the following lambda expressions, tell whether it is a valid lambda expression or not. If valid, write a functional interface method header it can match (use any arbitrary name for the method). If not, state what the error is.

(a) `() -> System.out.println("hello");`

Invalid, cannot use system.out in lambda expression

(b) `(a,b) -> (a || b) + ""`

Valid, boolean stuff (boolean a, boolean b);

(c) `String::length`

valid, String stuff (int length);

(d) `i -> return i*2;`

Invalid, return must be in braces

5. **Lambda Expressions (19 pts, 4+4+6+5)**

You are given the following class definition (assume all methods are correctly implemented):

```
public class Song {
   ...
   public Song(String name, String artist, String genre) { ... }
   public Song(String name, String artist) { ... }
   public String getGenre() { ... }
   public int copiesSold() { ... }
}
```

For each of the following, write NAMED and TYPED lambda expressions. In other words, LHS (left hand side) is a type and a variable name, and RHS (right hand side) is the lambda expression. For the type, use appropriate functional interfaces from the `java.util.function` and `java.util` packages. No need to write import statements.

(a) Get the genre of a song (do NOT use a method reference)

Function < String, Song > genre = Song :: getGenre()

(b) A method reference to create a Song instance instance with name and artist

BiFunction<String name, String artist> song = (s) -> {name, artist}

(c) A predicate for songs that are not of the genre "Pop" or "Rock", and have sold 10,000 or more copies. You may write named and typed supporting predicates, if needed.

predicate < song > genre != "Pop"
predicate < song > GetGenre() = "Rock"
predicate < song > copiesSold() >= 10000

Predicate < song > getgenre() != "Pop", "Rock" = copiesSold() >= 10000;

(d) An expression whose LHS variable can be passed as argument to the sort method of a List<Song> of songs, for sorting in ascending order of copies sold.

List < Song > = () -> copiesSold(), ascending