King Saud University

College of Computer and Information Sciences

Computer Science Department

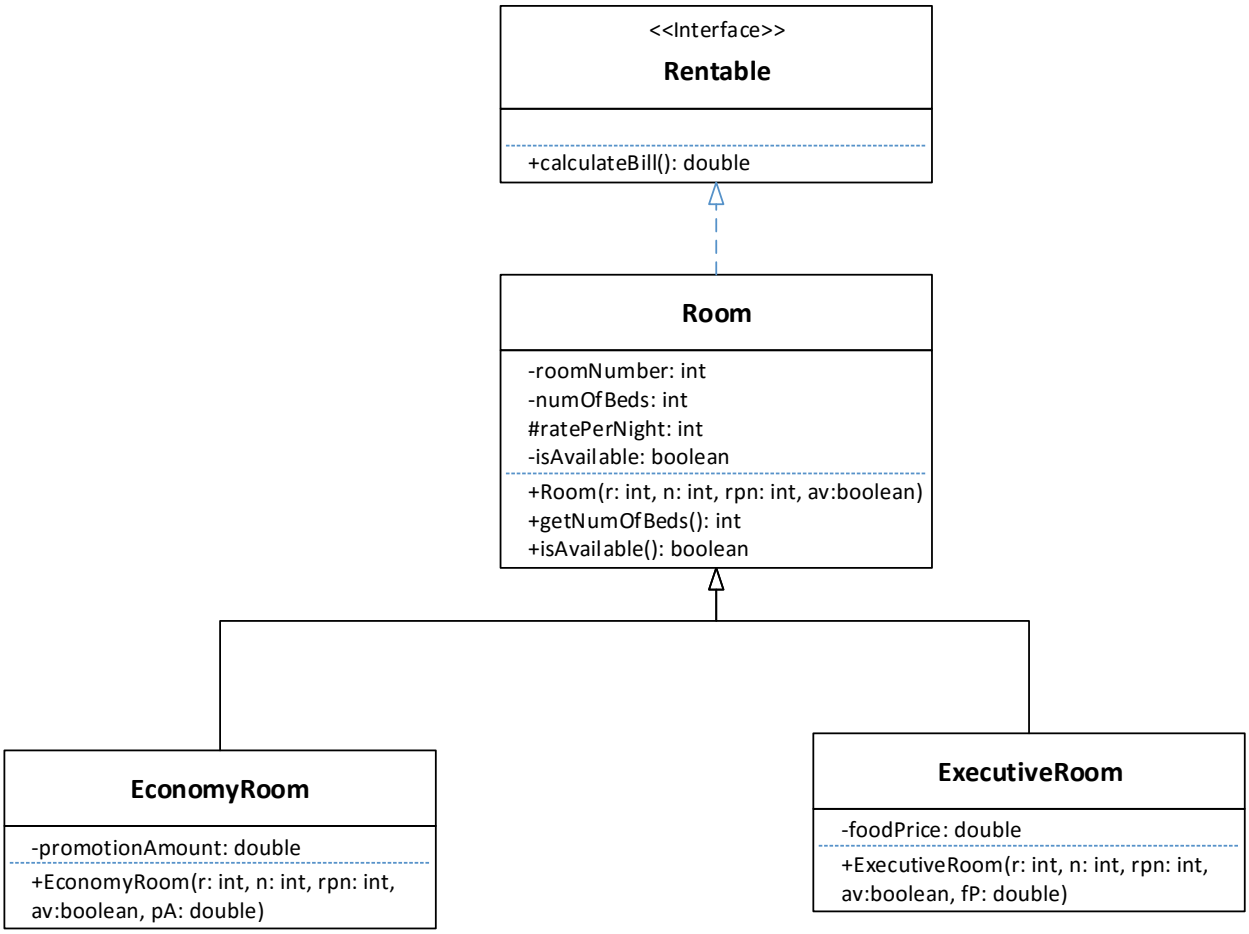
Course Code:	CSC 113
Course Title:	Computer Programming II
Semester:	Fall 2018
Exercises Cover Sheet:	**Final Exam**

Student Name:	
Student ID:	
Student Section No.	

Tick the Relevant	Computer Science B.Sc. Program ABET Student Outcomes	Question No. Relevant Is Hyperlinked	Covering %
X	**a) Apply knowledge of computing and mathematics appropriate to the computer science;**		
	b) Analyze a problem, and identify and define the computing requirements appropriate to its solution		
X	**c) Design, implement and evaluate a computer-based system, process, component, or program to meet desired needs;**		
X	**d) Function effectively on teams to accomplish a common goal;**		
	e) Understanding of professional, ethical, legal, security, and social issues and responsibilities;		
	f) Communicate effectively with a range of audiences;		
	g) Analyze the local and global impact of computing on individuals, organizations and society;		
	h) Recognition of the need for, and an ability to engage in, continuing professional development;		
X	**i) Use current techniques, skills, and tools necessary for computing practices.**		
	j) Apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices;		
	k) Apply design and development principles in the construction of software systems of varying complexity;		

Exercise 1:

Let's consider the following UML class diagram.



The interface Rentable:

- Methods:
  - ***calculateBill ()*:** This method will calculate the bill and throws an Exception as follows:
    - For ***ExecuttiveRoom***: Bill is calculated as (ratePerNight * 2) + foodPrice.
    - For ***EconomyRoom***: Bill is calculated as ratePerNight – promotionAmount. This method will throw an Exception if the promotionAmount is greater than ratePerNight.

The class Room:

- Attributes:
  - ***roomNumber:*** number of the room.
  - ***numOfBeds***: number of beds in the room.
  - ***ratePerNight***: rate of the room for one night.

- ***isAvailable:*** shows if the room is available or not.

- Methods:
  - ***Room (…)***: constructor.
  - ***getNumOfBeds()*:** this method returns the number of beds.
  - ***isAvailable():*** this method returns ***true*** if the room is available. It returns false otherwise.

The class EconomyRoom :

- Attributes:
  - ***promotionAmount***: offer or discounted amount on the room.
- Methods:
  - ***EconomyRoom (…):*** constructor.

The class ExecutiveRoom:

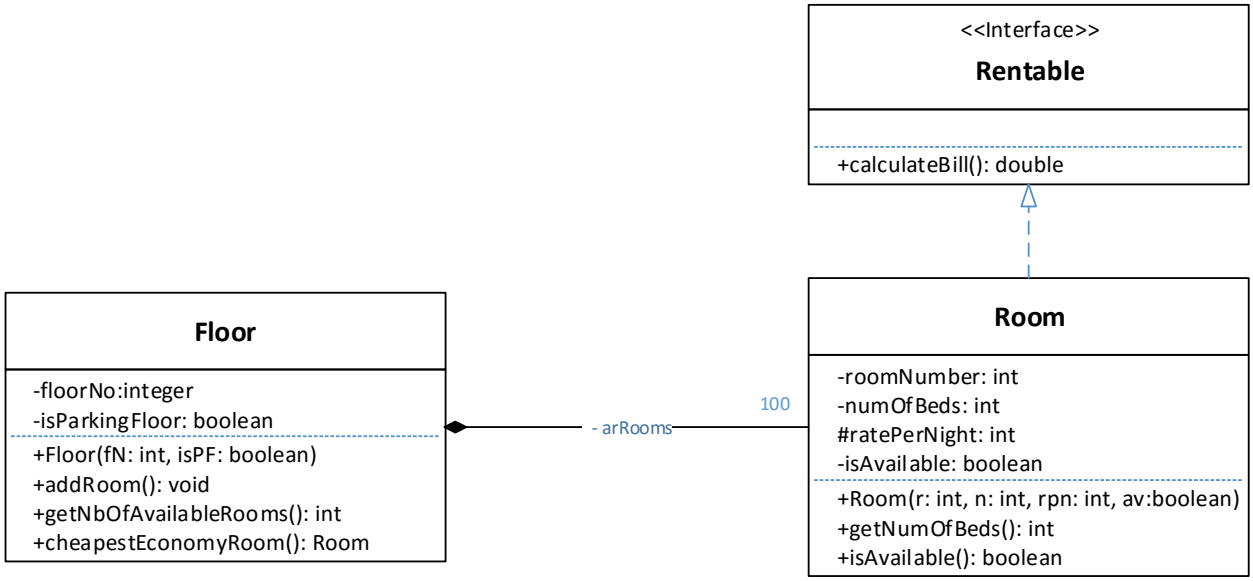
- Attributes:
  - ***foodPrice***: the price of the meal.
- Methods:
  - ***ExecutiveRoom (…):*** constructor.

QUESTION: Translate into Java code:

1. The interface ***Rentable***,
2. The class ***Room***
3. The class ***EconomyRoom.***

Exercise 2:

Let's consider the class ***Room*** and its subclasses as described in exercise 1.



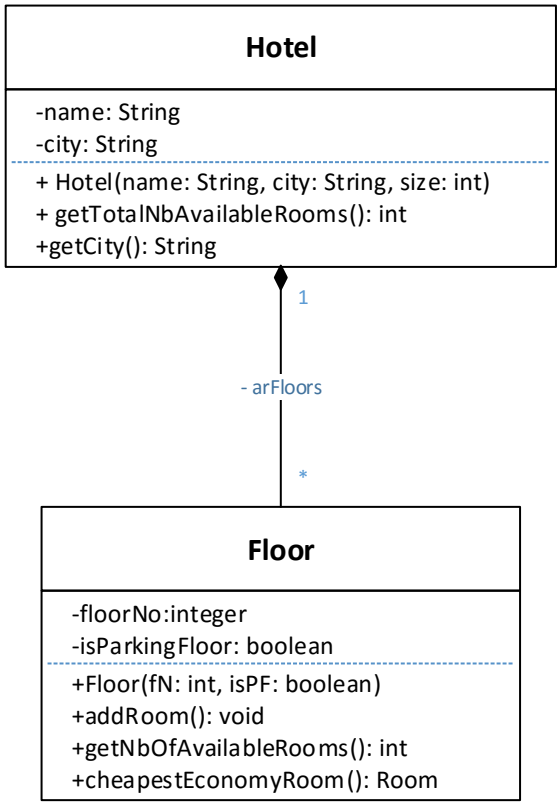
The class ***Floor:***

- Attributes:
  - ***floorNo***: the number of the Floor.
  - ***isParkingFloor***: shows if the floor is a parking floor or not.
- Methods:
  - ***Floor (…)***: constructor
  - ***addRoom (r: Room):*** this method adds the Room r in the Floor. This method throws an Exception with the message "This is a parking floor", if the floor is a parking floor. This method also throws an Exception with the message "Floor is full", if the floor is full.
  - ***getNbOfAvailableRooms():*** This method will return the number of available rooms in the floor. This method throws an Exception with the message "No rooms" if the floor is a parking floor.
  - ***cheapestEconomyRoom***(): This method will return a room having the lowest price (bill).

QUESTION: Translate into Java code the class ***Floor.***

Exercise 3:

Let's consider the class ***Floor*** as described in exercise 2.



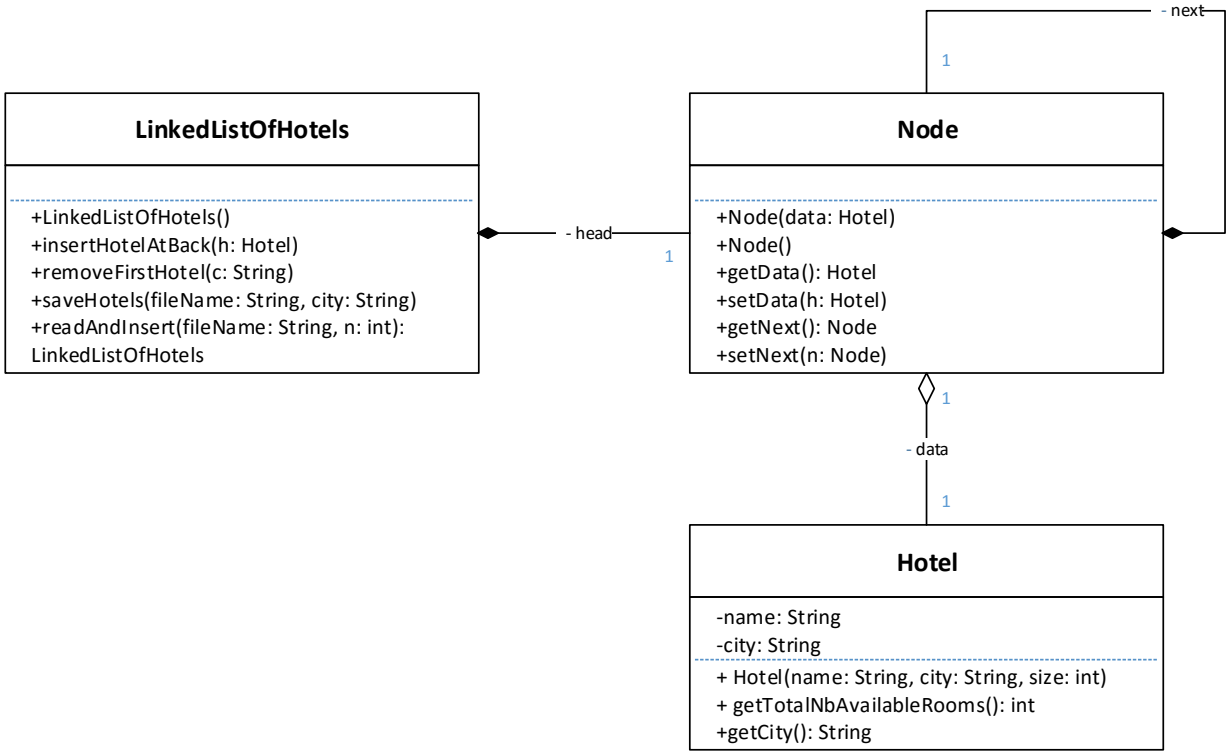
The class ***Hotel:***

- Attributes:
  - ***name***: the name of the hotel.
  - ***city***: city name of the hotel.
- Methods:
  - ***Hotel (…)***: constructor
  - ***getTotalNbAvailableRooms():*** this method returns the total number of available rooms in the hotel.
  - ***getCity():*** This method will return the name of the city of the hotel.

QUESTION: Translate into Java code the class ***Hotel.***

Exercise 4:

Let's consider the class ***Hotel*** as described in exercise 3.



The class ***Node:***

- Attributes.
- Methods:
  - ***Node (…)***: constructors
  - ***getData():*** this method return the data.
  - ***getNext():*** This method returns the next.
  - ***setData():*** this method sets the value of the data.
  - ***setNext():*** This method sets the value of next.

The class ***LinkedListOfHotels:***

- Attributes.
- Methods:
  - ***LinkedListOfHotels (…)***: constructor
  - ***insertHotelAtBack(h***: Hotel): This method will add a new hotel at the end of the linked list.
  - ***removeFirstHotel(c***: String): This method will remove the first hotel from the linked list with the city name ***c***.

- ***saveHotels(fileName***: String, ***c***: String): This method will save all the hotels of city c into the file fileName.
- ***readAndInsert(filename***: String, ***n***: int): This method returns a linked list that contains all hotels it reads from the file ***filename*** with at least ***n*** available rooms.

QUESTION: Translate into Java code the class ***LinkedListOfHotels.***

////////////////////////////////////

Exercise 1:

```
public interface Rentable {  …….. 1          ……../3
      public double calculateBill() throws Exception; …….. 1+1
}
```

```
public abstract class Room implements Rentable { …….. 1+1    ……../7
      private int roomNumber;
      private int numOfBeds;
      protected int ratePerNight;
      private boolean available;

      public Room(int r, int n, int rpn, boolean b)  { …….. 1
            this.roomNumber = r;
            this.numOfBeds = n;
            this.ratePerNight = rpn;
            this.available = b;
      }

      public Room(Room r)  { …….. 2
            this.roomNumber = r.roomNumber;
            this.numOfBeds = r.numOfBeds;
            this.ratePerNight = r.ratePerNight;
            this.available = r.available;
      }


      public int getNumOfBeds() { …….. 1
            return numOfBeds;
```

```
	}

	public boolean isAvailable() { ……….. 1
		return available;
	}
}
```

```
public class EconomyRoom extends Room {    …….. 1          ……../8
      private double promotionAmount;

      public EconomyRoom(int r, int n, int rpn, boolean b, double d) {
            super(r, n, rpn, b); …….. 1
            this.promotionAmount = d;
      }

      public EconomyRoom(EconomyRoom er) { …….. 1 ……../2
            super(er); …….. 1
            this.promotionAmount = er.promotionAmount;
      }

      public double calculateBill() throws Exception { …… 1  ……../4
            if (promotionAmount > ratePerNight) …….. 1
                  throw new Exception("Error...!"); …….. 1
            return (ratePerNight - promotionAmount); …….. 1
      }
}

Exercice 2:
public class Floor { ……../37
      private int floorNo;
      private boolean isParking;

      private Room arRooms[]; …….. 1
      int nbRooms; …….. 1

      public Floor(int fn, boolean b) { ……../2
            floorNo = fn;
            isParking = b;

            arRooms = new Room[100]; …….. 1
            nbRooms = 0;             …….. 1
      }

      public Floor(Floor f) { ……../6
            floorNo = f.floorNo;
            isParking = f.isParking;

            arRooms = new Room[f.arRooms.length]; …….. 1
            nbRooms = 0;                          …….. 1

            for (int i=0; i < f.nbRooms; i++) { …….. 1
                  try {                       …….. 1
                        addRoom(f.arRooms[i]); …….. 1
                  }
                  catch(Exception e) {        …….. 1
                              System.out.println(e.getMessage());
                  }
            }
      }
```

```
public void addRoom(Room r) throws Exception {  …….. 1   ……../11
        if (isParking) …….. 1
            throw new Exception("This is a parking floor."); …….. 1

        if (nbRooms < arRooms.length) { …….. 1

            if (r instanceof EconomyRoom) …….. 1
               arRooms[nbRooms] = new EconomyRoom((EconomyRoom)r); …1+1
            else
               arRooms[nbRooms] = new …1+1
                                      ExecutiveRoom((ExecutiveRoom)r);

            nbRooms++; …….. 1
        }
        else
            throw new Exception("Floor is full."); …….. 1
    }

    public int getNbOfAvailableRooms() throws Exception { …….. 1 ……../8
        int n = 0;  …….. 1
        if (isParking) …….. 1
            throw new Exception("No rooms."); …….. 1

        for (int i = 0; i < nbRooms; i++)  …….. 1
            if (arRooms[i].isAvailable()) …….. 1
               n++; …….. 1

        return n; …….. 1
    }

    public Room cheapestEconomyRoom() { ……../8
        Room min = null;

        for (int i=0; i < nbRooms; i++) {  …….. 1
            if (arRooms[i] instanceof EconomyRoom) …….. 1 {
               try { …….. 1
                  if (min == null || …….. 1+1
                     arRooms[i].calculateBill() < min.calculateBill())
                              min = arRooms[i]; …….. 1
               }
               catch(Exception e) { …….. 1
                  System.out.println(e.getMessage());
               }
            }
        }

        return min; …….. 1
    }
}
```

Exercice 3:

```
public class Hotel implements Serializable {    ……../11
      private String name, city;

      private Floor arFloors[]; …….. 1
      private int nbFloor;  …….. 1

      public Hotel(String s, String c, int size) { ……../2
            name = s;
            city = c;

            arFloors = new Floor[size]; …….. 1
            nbFloor = 0;  …….. 1
      }

      public int getTotalNbOfAvailableRooms() { ……../6
            int n= 0;          …….. 1

            for (int i = 0; i < nbFloor; i++) { …….. 1
                  try {                        …….. 1
                        n += arFloors[i].getNbOfAvailableRooms(); …….. 1
                  }
                  catch(Exception e) { …….. 1
                        System.out.println(e.getMessage());
                  }
            }

            return n; …….. 1
      }

      public String getCity() { ……../1

            return city;
      }

}
```

```
Exercise 4:
public class LinkedListOfHotels {   ……../44
      private Node head;       …….. 1

      public LinkedListOfHotels() {
            head = null;       …….. 1
      }

      public void insertHotelAtBack(Hotel h) {  ……../7
            Node current = new Node(h);    …….. 1

/*          Node current = new Node();
            current.setData(h);
*/

            if (head == null) {  …….. 1
                  head = current; …….. 1
            }
            else {
                  Node tail = head; …….. 1
                  while (tail.getNext() != null)       …….. 1
                        tail = tail.getNext();   …….. 1

                  tail.setNext(current);   …….. 1
            }
      }

      public void removeFirst(String c) {  ……../11
            Node current = head;     …….. 1
            Node previous = null;    …….. 1

            while (current != null) {      …….. 1
                  if (current.getData().getCity().equals(c)) {  …….. 1+1
                        if (previous == null)                  …….. 1
                              head = current.getNext();        …….. 1
                        else
                              previous.setNext(current.getNext()); …….. 1

                        return;      …….. 1
                  }
                  else {
                        previous = current;      …….. 1
                        current = current.getNext();   …….. 1
                  }
            }
      }
```

```
public void saveHotels(String fileName, String c) throws IOException { ...../10
            File f = new File(fileName);  ..... 1
            FileOutputStream fos = new FileOutputStream(f);        ..... 1
            ObjectOutputStream objF = new ObjectOutputStream(fos); ..... 1

            Node current = head; ..... 1
            while (current != null) {      ..... 1
                  if (current.getData().getCity().equals(c)) ..... 1+1
                        objF.writeObject(current.getData()); ..... 1+1
            }

            objF.close(); ..... 1
      }

      public LinkedListOfHotels readAndInsert(String fileName, int n) ...../14
                              throws IOException {
            File f = new File(fileName);  ..... 1
            FileInputStream fis = new FileInputStream(f);    ..... 1
            ObjectInputStream objF = new ObjectInputStream(fis); ..... 1

            LinkedListOfHotels res = new LinkedListOfHotels(); ..... 1
            Hotel h; ..... 1
            try { ..... 1
                  while (true) {     ..... 1
                        h = (Hotel) objF.readObject();       ..... 1+1

                        if (h.getTotalNbOfAvailableRooms() >= n ) ..... 1
                              res.insertHotelAtBack(h); ..... 1
                  }
            }
            catch(EOFException e) { ..... 1
                  objF.close(); ..... 1
            }
            catch (ClassNotFoundException e) {
                  e.printStackTrace();
            }

            return res; ..... 1
      }
}
```