



King Saud University

College of Computer and Information Sciences

Computer Science Department

Course Code:

CSC 113

Course Title:

Computer Programming II

Semester:

Spring 2020

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

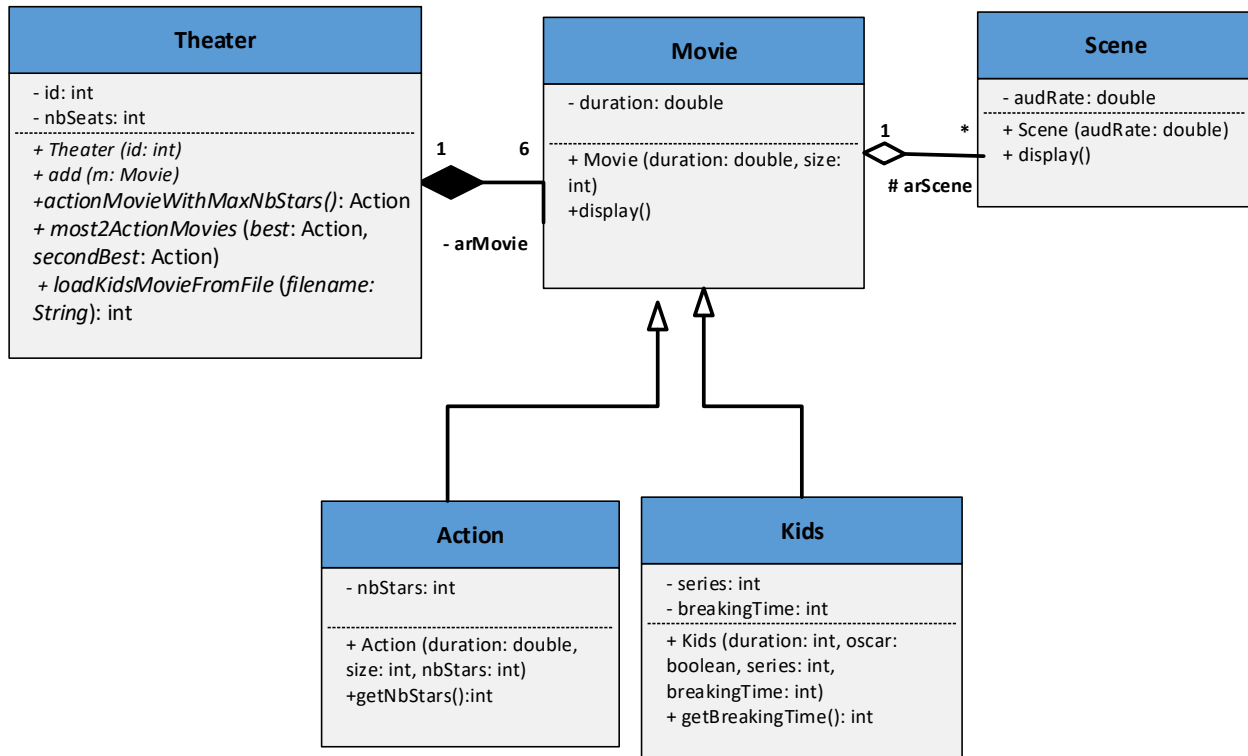
i) Use current techniques, skills, and tools necessary for computing practices.

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 class **Scene**:

- Attributes: **audRate**: The audience rate of the scene.
- Methods:
 - **Scene (audRate: double)**: Constructor
 - **display()**: displays all attributes of the Scene.

The class **Movie**:

- Attributes: **duration**: The duration time of the Movie.
- Methods:
 - **Movie (duration: double, size: int)**: Constructor
 - **display()**: displays all attributes of the Movie.

The class **Action**:

- Attributes: **nbStars**: The number of stars (actors) of the Action movie.
- Methods:
 - **Action (duration: double, size: int, nbStars: int)**: Constructor.
 - **getNbStars ()**: returns the number of stars (actors) of the Action movie.

QUESTION:

1. Implement the constructors of the class **Movie**,
2. Implement the constructors of the class **Action**.

Exercise 2:

Let's consider the same UML class diagram described above in Exercise 1. Let's consider that the class **Movie** is an abstract class.

The class Theater:

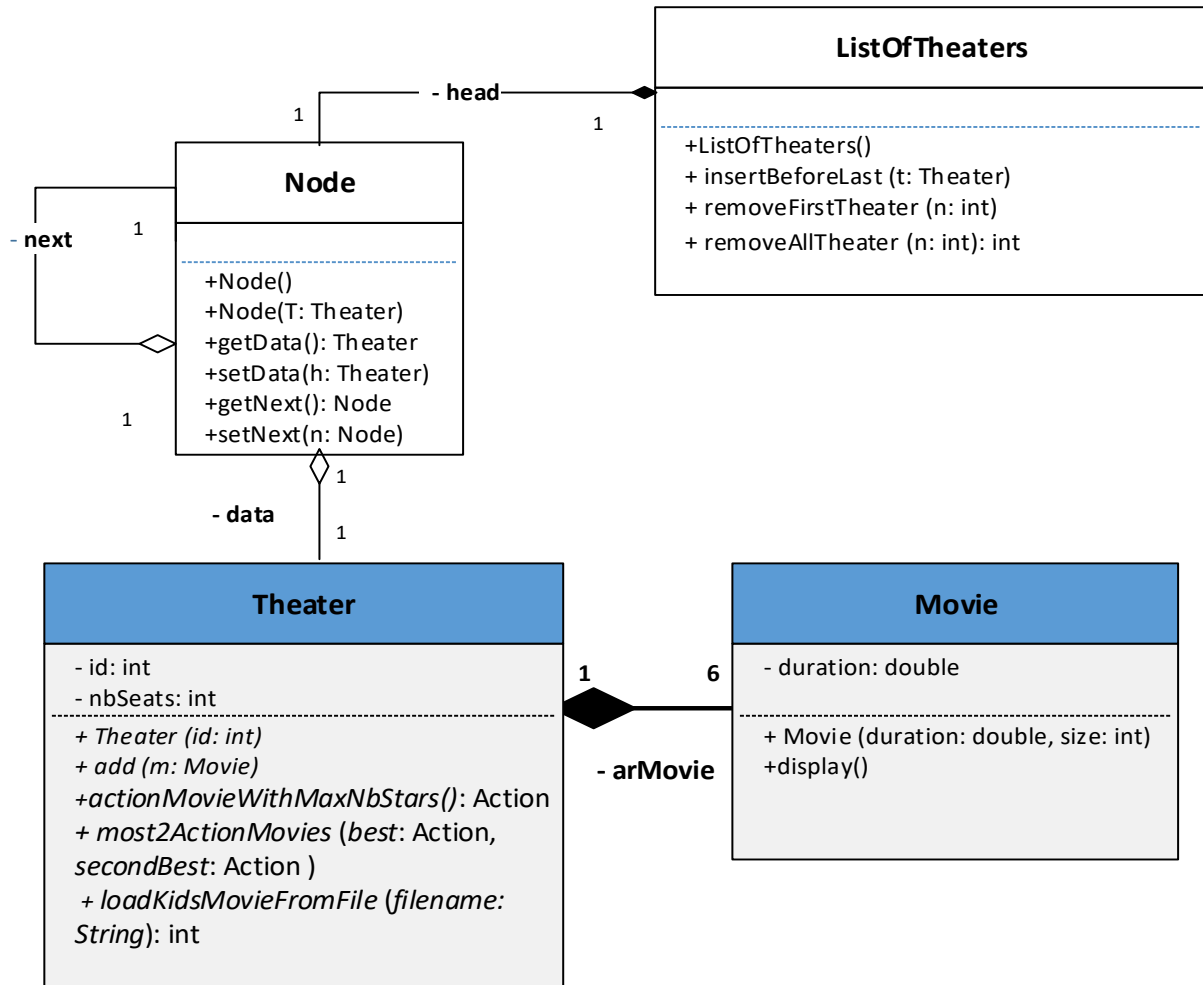
- Attributes:
 - **id**: The identifier of the Theater.
 - **nbSeats**: The number of seats of the Theater.
- Methods:
 - **Theater (ID: int)**: constructor.
 - **add(m: Movie)**: This method adds the Movie **m** to the Theater. It returns an exception of type **ArrayIndexOutOfBoundsException** if the array of Movies is full.
 - **actionMovieWithMaxNbStars()**: It return the Action movie that involves the maximum number of stars (actors).
 - **most2ActionMovies (best: Action, secondBest: Action)**: This method receives two parameters **best** and **secondBest** both of type Action movies. This method:
 - searches (i) the Action movie that has the maximum number of stars and (ii) the Action movie that has the second maximum number of stars;
 - and stores them in the parameters **best** and **secondBest** respectively.
 - **loadKidsMovieFromFile(filename: String)**: This method reads Kids movies from a file of Movie objects named **filename** and adds them to the Theater. It returns the number of Kids movies added to the Theater.
 - **getNbSeats ()**: returns the number of seats of the Theater.

QUESTION: Implement using Java the following methods of the class **Theater**:

1. **add(m: Movie)**
2. **actionMovieWithMaxNbStars(): Action**
3. **most2ActionMovies(best: Action, secondBest: Action)**
4. **loadKidsMovieFromFile(filename: String): int**

Exercise 3:

Let's consider the same class *Theatre* as described in Exercise 2.



The class ListOfTheaters:

- Methods:
 - **ListOfTheaters ()**: constructor.
 - **insertBeforeLast (t: Theater)**: This method will add a new theater before the last element of the linked list.
 - **removeFirstTheater (n: int)**: This method will remove the first theater in the list having a number of seats less than *n*.
 - **removeAllTheater (n: int)**: This method will remove all theater having a number of seats less than *n*. It returns the number of deleted theaters.

QUESTION: Implement using Java the following methods of the class *ListOfTheaters*:

1. **insertBeforeLast (t: Theater)**
2. **removeFirstTheater (n: int)**
3. **removeAllTheater (n: int): int**

Solution

Exercise One:

Class *Movie*

```
public class Movie
{
    private double duration;
    protected Scene[] arScene;
    protected int nb;

    public Movie(double duration, int size)
    {
        this.duration = duration;
        arScene = new Scene[size];
        nb = 0;
    }

    public Movie (Movie m)
    {
        this.duration = m.duration;
        arScene = new Scene[m.arScene.length];
        this.nb = m.nb;
        for (int i=0 ; i < m.nb ; i++)
            this.arScene[i] = m.arScene[i];
    }

    public void display()
    {
        System.out.println("Duration: " + duration);
        for(int i =0; i<arScene.length; i++)
            arScene[i].display();
    }
}
```

Class *Action*

```
public class Action extends Movie
{
    private int nbStars;

    public Action(double duration, int size, int nbStars)
    {
        super(duration, size);
        this.nbStars = nbStars;
    }

    public Action(Action a)
    {
        super(a);
        this.nbStars = a.nbStars;
    }

    public int getNbStars()
    {
        return nbStars;
    }
}
```

Exercise Two:

Class *Theater*

```
public class Theater
{
    private int id;
    private int nbSeats;
    private Movie[] arMovie;
    private int nb;

    public Theater(int id)
    {
        this.id = id;
        nbSeats = 0;
        arMovie = new Movie[6];
        nb = 0;
    }

    public void add(Movie m) throws ArrayIndexOutOfBoundsException
    {
        if (nb >= arMovie.length)
            throw new
                ArrayIndexOutOfBoundsException("Array is full");
        if (m instanceof Action)
            arMovie[nb] = new Action((Action) m);
        else
            arMovie[nb] = new Kids((Kids) m);
        nb++;
    }
}
```



```

public Action actionMovieWtihMaxNbStars()
{
    Action result = null;

    for (int i = 0; i < nb; i++)
    {
        if (arMovie[i] instanceof Action)
            if (result == null || ((Action)
arMovie[i]).getNbStars() > result.getNbStars())
                result = (Action) arMovie[i];
    }
    return result;
}

```

```

public void most2ActionMovies(Action best, Action secondBest)
{
    best = actionMovieWithMaxNbStars();

    int secondMaxnbStars = 0;

    int j = 0;

    for (int i = 0; i < nb; i++)
    {
        if (arMovie[i] instanceof Action)
        {
            if (((Action) arMovie[i]).getNbStars() >
secondMaxnbStars)
            {
                if (((Action)
arMovie[i]).getNbStars() != best.getNbStars())
                {
                    j = i;
                    secondMaxnbStars = ((Action)
arMovie[i]).getNbStars();
                }
            }
        }
        secondBest = (Action) arMovie[j];
    }
}

```

```

    public int loadKidsMovieFromFile(String filename) throws
IOException
    {
        File f = new File(filename);
        FileInputStream fin = new FileInputStream(f);
        ObjectInputStream obj = new ObjectInputStream(fin);

        int count = 0;

        Movie m;

        try
        {
            while (true)
            {
                m = (Movie) obj.readObject();
                if (m instanceof Kids)
                    add((Kids) m);
            }
        }
        catch (EOFException e)
        {
        }

        catch (ClassNotFoundException e)
        {
        }

        catch (ArrayIndexOutOfBoundsException e)
        {
        }

        obj.close();
        return count;
    }
}

```

Exercise Three:

Class *ListOfTheaters*

```
public class ListOfTheaters {
    private Node head;

    public ListOfTheaters()
    {
        head = null;
    }

    public void insertBeforeLast(Theater t)
    {
        Node newNode = new Node(t);

        if (head == null) {
            head = newNode;
            return;
        }
        else if (head.getNext() == null) {
            head.setNext(newNode);
            newNode.setNext(newNode.getNext());
            return;
        }

        Node p = head;
        Node q = head.getNext();
        while (q != null)
        {
            p = q;
            q = q.getNext();
        }
        p.setNext(newNode);
        newNode.setNext(q);
        return;
    }
}
```

```

public void removeFirstTheater(int n)
{
    Node current = head;
    Node previous = null;

    while (current != null)
    {
        if (current.getData().getNbSeats() < n)
        {
            if (previous == null) {
                head = current.getNext();
                return;
            }
            else
            {
                previous.setNext(current.getNext());
                return;
            }
        }
        else
        {
            previous = current;
            current = current.getNext();
        }
    }
}

```

```

public int removeAllTheater(int n)
{
    int counter = 0;
    Node current = head.getNext();
    Node previous = head;

    if (head != null && head.getData().getNbSeats() < n)
    {
        head = head.getNext();
        counter++;
    }

    while (current != null)
    {
        if (current.getData().getNbSeats() < n)
        {
            previous.setNext(current.getNext());
            counter++;
        }
        previous = current;
        current = current.getNext();
    }
    return counter;
}

```