King Saud University
College of Computer and Information Sciences
CSC111
Lab Final Exam Time 1:40

Write a Java program that shows movie times in a cinema. Here are the UML diagrams:

Film	3 marks
- length: int - name: String - startH: int - startM: int	1
+ Film () + Film (length: int, name:String, startH: int, startM:int) + setters + getters	0.5 0.5 0.5 0.5

Cinema	12 marks
- films[ ]: Film - nFilms: integer + MAX_SIZE: public static final integer	1
+ main() + Cinema () + getNFilms(): int + addFilm(f: Film) : void + removeFilm(index: int): void + displayFilm(name:String): void + displayShowTimes(hour: int, min: int):void + displayAll():void	4 0.5 0.5 1 1 1 2 1

The Class: Film

As shown in the UML diagram, write the class Film that has the attributes: length, name, startH (starting hour) and startM (starting minute). (for example a movie that starts 19:30 will have startH=19 and startM=30). Last movie starts before midnight.

- **length**: Length of the movie in minutes (int).
- **name**: Name of the movie (String).
- **startH**: The hour in which the movie starts (int).
- **startM**: The minute in which the movie starts (int).

The methods of this class are:

- **Film**(): A default constructor.
- **Film**(length,name,startH,startM): A constructor that initializes new film with the initial values from the user.

- **Setter methods** (one for each): That sets the values for: length,name,startH,startM
- **Getter Methods** (one for each): That returns the values for: length,name,startH,startM

The Main Class: Cinema

The main class is class **Cinema** which is the class that you are going to use to test your program. This class contains an array of objects **films[ ]** that contains all the films showing in a cinema. It also contains the variable **nFilms** that represents the currate number of films stored in the array of objects **films[ ]**.

This class contains main method as well as other methods. Note that the maximum number of films in the array is 100 (hint: **MAX_SIZE** is a constant).

The methods of this class are:

- **Cinema**(): A constructor that initializes the attributes and creates an array of films of size MAX_SIZE
- **getNFilms**: Returns the currate number of films stored in the array.
- **addFilm**: This method adds a film to the array. If it is not possible to add the film, you should print the error message: "**Adding Film Not Possible!**".
- **removeFilm**: This method removes the film at the given **index** by deleting that film and replacing it with the last film in the array **films**.
- **displayFilm**: Displays all film details for film with given **name.**
- **displayShowTimes:** Display all films showing after the given time. (For example: all the films that start on or after 20:10)
- **displayAll**: Display all films showing in the cinema.
- **main**: the main method does the following:

  1. Create a **Cinema** object.
  2. Ask the user to enter film information. Keep reading and storing films information in array films until user enters "stop" as film name.
  3. Display a small menu with following five options:
     1. Display film,
     2. Display all films after your time,
     3. Display all films,
     4. Remove film,
     5. Exit the program.
  4. Do options 1 – 3 at least once
  5. Delete a movie
  6. Display all movies
  7. Exit

Sample run:

```
Welcome to Riyadh Cinema ticketing System. Please enter the films, enter "stop" to exit
Please enter Film name, length and starting time: film00 100 19 20
Please enter Film name, length and starting time: film01 120 19 30
Please enter Film name, length and starting time: film02 90 15 50
Please enter Film name, length and starting time: film03 180 21 45
Please enter Film name, length and starting time: film04 90 22 15
Please enter Film name, length and starting time: film05 150 19 45
Please enter Film name, length and starting time: stop
Please enter your option:
1) Display film          2) Display All films after your time          3) Display all films
4) Remove film           5) Exit Program
> 1
Please enter the film name you want to display: film01
the Film film01 is 120 minutes long. It starts at 19:30
Please enter your option:
1) Display film          2) Display All films after your time          3) Display all films
4) Remove film           5) Exit Program
> 2
Please enter the time you want to see a movie: 19 30
the Film film01 is 120 minutes long. It starts at 19:30
the Film film03 is 180 minutes long. It starts at 21:45
the Film film04 is 90 minutes long. It starts at 22:15
the Film film05 is 150 minutes long. It starts at 19:45
Please enter your option:
1) Display film          2) Display All films after your time          3) Display all films
4) Remove film           5) Exit Program
> 4
Please enter the index of the film you want to delete: 3
Please enter your option:
1) Display film          2) Display All films after your time          3) Display all films
4) Remove film           5) Exit Program
> 4
Please enter the index of the film you want to delete: 2
Please enter your option:
1) Display film          2) Display All films after your time          3) Display all films
4) Remove film           5) Exit Program
> 3
the Film film00 is 100 minutes long. It starts at 19:20
the Film film01 is 120 minutes long. It starts at 19:30
the Film film04 is 90 minutes long. It starts at 22:15
the Film film05 is 150 minutes long. It starts at 19:45
Please enter your option:
1) Display film          2) Display All films after your time          3) Display all films
4) Remove film           5) Exit Program
> 2
Please enter the time you want to see a movie: 22 10
the Film film04 is 90 minutes long. It starts at 22:15
Please enter your option:
1) Display film          2) Display All films after your time          3) Display all films
4) Remove film           5) Exit Program
> 2
Please enter the time you want to see a movie: 23 10
Please enter your option:
1) Display film          2) Display All films after your time          3) Display all films
4) Remove film           5) Exit Program
> 5
Goodbye!
```

Write a Java program that manages flights in an airport. Here are the UML diagrams:

Flight	3 points
- id: int - seats: int - dest: String - status: String - price: double	1
+ Flight() + Flight(id: int, seats: int, dest: String, status: String, price: double) + setters + getters	0.5 0.5 0.5 0.5

Airport	8 points
- flights[]: Flight - numFlights: int + MAX_SIZE: public static final int	1
+ Airport() + getNumFlights(): int + addFlight(f: Flight): void + removeFlight(i: int): void + displayFlight(i: int): void + findMinPriceTo(destination: String): int + buyTicket(i: int): boolean	1 0.5 1 1 0.5 1.5 1.5

TestAirport	4 points
+ main(): void	4

The Class: Flight

As shown in the UML diagram, write the class Flight that has the attributes:

- ***id***: the id of the flight
- ***seats***: the number of empty seats in the flight available for sale.
- ***dest***: the name of the destination city.
- ***status***: represents availability of the flight, either "**Available**" or "**Departed**".
- ***price***: ticket price.

The methods of this class are:

- **Flight()**: A default constructor that sets id, seats, and price to -1 and dest and status to "
- **Flight**(id,seats,dest,status,price): A constructor that initializes new flight with the init
  the user.

- **Setter methods** (one for each): That sets the values for: id, seats, dest, status, and price.
- **Getter Methods** (one for each): That returns the values for: id, seats, dest, status, and price.

The Class: Airport

The class **Airport** contains an array of objects **flights[ ]** that contains all the flights that will depart from the airport. It also contains the variable **numFlights** that represents the current number of flights stored in the array of objects **flights[ ]**.

Note that the maximum number of apartments in the list is 100 (hint: **MAX_SIZE** constant).

The methods of this class are:

- **Airport**(): a constructor that initializes the attributes and creates an array of apartments of size 100.
- **getNumFlights**: returns the current number of flights stored in the array.
- **addFlight**: this method will add a flight. If it is not possible to add the flight, you should print an error message "**ERROR Adding**".
- **removeFlight**: remove the flight with index 'i' and put the last flight in its place. Otherwise print "**ERROR Removing**".
- **displayFlight**: displays the details for the flight with index 'i' in the array, otherwise print "**ERROR Printing**". It should display the flight information as follows:

```
The flight no. 1 to Makkah is Available. Ticket price is 600 SR. 100 seats left.
```

- **findMinPriceTo**: returns the index of the cheapest flight ticket to the given "destination". If no such flight exist then return -1. (Note: status should be Finally **Available**")
- **buyTicket**: if the flight status at index 'i' is "**Available**" and there is at least one seat empty, then reduce the number of seats by 1 and return true. Otherwise, return false. Print "**ERROR Buying**" only if 'i' doesn't exist .

The Main Class: TestAirport

- **main**: the main method does the following:
  1. Create an **Airport** object.
  2. (1 point) Ask the user to enter flights information. Keep reading and storing flights information in array flights until user enters (-1) as id of flight.
  3. (1 point) Ask the user to enter the destination he want to go to and display the flight with the cheapest ticket. Or print "No flights to destination" if there was none.
     a. (0.5 points) If there was a flight, ask the user if he wasn't to buy a ticket.
     b. (0.5 points) If "yes", show the user if he was able to buy a ticket or not.
  4. (1 point) Finally, display all flights.

Sample run:

```
Enter flight id (-1 to exit), seats, distention, status, and price:
-1
```

Sample run:

```
Enter flight id (-1 to exit), seats, distention, status, and price:
111 200 Jeddah Available 500
Enter flight id (-1 to exit), seats, distention, status, and price:
-1
What is your distention? Makkah
No flights to Makkah
All flights:-
The flight no. 111 to Jeddah is Available. Ticket price is 500 SR. 200 seats left.
```

Sample run:

```
Enter flight id (-1 to exit), seats, distention, status, and price:
111 200 Jeddah Available 500
Enter flight id (-1 to exit), seats, distention, status, and price:
222 0 Jeddah Available 250
Enter flight id (-1 to exit), seats, distention, status, and price:
333 200 Dammam Available 500
Enter flight id (-1 to exit), seats, distention, status, and price:
444 200 Jeddah Departed 300
Enter flight id (-1 to exit), seats, distention, status, and price:
-1
What is your distention? Jeddah
Here is the cheapest flight to Jeddah
The flight no. 222 to Jeddah is Available. Ticket price is 250 SR. 0 seats left.
Want to buy a ticket in that flight? yes
Couldn't buy ticket. Either the flight is not Available or it's full.
All flights:-
The flight no. 111 to Jeddah is Available. Ticket price is 500 SR. 200 seats left.
The flight no. 222 to Jeddah is Available. Ticket price is 250 SR. 0 seats left.
The flight no. 333 to Dammam is Available. Ticket price is 500 SR. 200 seats left.
The flight no. 444 to Jeddah is Departed. Ticket price is 300 SR. 200 seats left.
```

Sample run:

```
Enter flight id (-1 to exit), seats, distention, status, and price:
111 200 Jeddah Available 500
Enter flight id (-1 to exit), seats, distention, status, and price:
222 200 Jeddah Available 400
Enter flight id (-1 to exit), seats, distention, status, and price:
333 200 Dammam Available 500
Enter flight id (-1 to exit), seats, distention, status, and price:
444 200 Jeddah Available 1300
Enter flight id (-1 to exit), seats, distention, status, and price:
-1
What is your distention? Jeddah
Here is the cheapest flight to Jeddah
The flight no. 222 to Jeddah is Available. Ticket price is 400 SR. 200 seats left.
Want to buy a ticket in that flight? yes
Done.
All flights:-
The flight no. 111 to Jeddah is Available. Ticket price is 500 SR. 200 seats left.
The flight no. 222 to Jeddah is Available. Ticket price is 400 SR. 199 seats left.
The flight no. 333 to Dammam is Available. Ticket price is 500 SR. 200 seats left.
The flight no. 444 to Jeddah is Available. Ticket price is 1300 SR. 200 seats left.
```

a Java program that manages players in a game clan. Here are the UML diagrams:

Player	3.5 points
- name: String - age: int - rank: int - server: String	1
+ Player() + Player(name: string, age: int, rank: int, server: String) + setters + getters	0.5 1 0.5 0.5

Clan	7.5 points
- player[]: Player - numPlayer: int + MAX_SIZE: public static final int	1
+ Clan() + getNumPlayers(): int + addPlayer(a: Player): void + removePlayer(n: string): void + displayPlayer(n: string): void + getAveAge(server: String): double + findLeastRank(): int	1 0.5 1 1 0.5 1 1.5

TestClan	4 points
+ main(): void	4

The Class: Player

As shown in the UML diagram, write the class Player that has the attributes:

- **name**: the name of the athlete.
- **age**: the athlete's age.
- **rank**: the game rank of the player so far.
- **server**: the game server from where player is playing.

The methods of this class are:

- **Player()**: A default constructor that sets id, age, and distance to -1 and server to "NA".
- **Player**(name, age, rank, server): A constructor that initializes new player with the initial values from the user.
- **Setter methods** (one for each): That sets the values for: name, age, rank, and server.
- **Getter Methods** (one for each): That returns the values for: name, age, rank, and server.

The Class: Clan

lass **Clan** contains an array of objects **players[ ]** that contains all the players that are in the clan. It also ins the variable **numPlayers** that represents the current number of players stored in the array of objects **ers[ ]**.

that the maximum number of elements in the list is 100 (hint: **MAX_SIZE** constant).

methods of this class are:

- **Clan()**: a constructor that initializes the attributes and creates an array of size 100
- **getNumPlayers**: returns the current number of players stored in the array.
- **addPlayer**: this method will add a player. If it is not possible to add the player, you should print an error message "**ERROR Adding**".
- **removePlayer**: remove the player with index 'i' and put the last player in its place. Otherwise print "**ERROR Removing**".
- **displayPlayer**: displays the details for the player with index 'i' in the array, otherwise print "**ERROR Printing**". It should display the player information as follows:

```
The Player Name is Abbas, age is 20 and has Rank 343
```

- **getAvgAge**: returns the average age of all players from a given "server". If no such players exists, then return -1.
- **findLeastRank**: return the details of the player with the least rank (higher the rank number least is the rank). If more than one has the same rank, return the first one. If no players exist, return -1.

The Main Class: TestClan

- **main**: the main method does the following:

1. Create a **Clan** object.
2. (1 point) Ask the user to enter player's information. Keep reading and storing player's information in array players[] until user enters (-1) as the player's id.
3. (0.5 point) Ask the user to enter a server, then print their average age. If no player with the given server exist, then print "No one from this server participated."
4. (1 point) Display the player with the least rank.
5. Ask the user if they want to remove him/her from the clan:
   a. (0.5 points) If "yes", remove the player from the Clan object.
6. (1 point) Finally, display all Players.

ple run:

```
ter Player id (-1 to exit), age, rank, and server:
bas 20 343 Middle East
ter Player id (-1 to exit), age, rank, and server:
1
nter a server: Middle East
he average age for players from Middle East server is 20
he player with the least ranking is:
The Player Name is Abbas, age is 20 and has Rank 343
Do you want to remove this player (y/n)? y
This clan has no players.
```

Sample run:

```
Enter Player id (-1 to exit), age, rank, and server:
Abbas 20 343 Middle East
Enter Player id (-1 to exit), age, rank, and server:
Ali 24 454 North America
Enter Player id (-1 to exit), age, rank, and server:
Ahmed 40 345 South America
Enter Player id (-1 to exit), age, rank, and server:
Osman 36 585 South America
Enter Player id (-1 to exit), age, rank, and server:
-1
Enter a server: Middle East
The average age for players from Middle East server is 20
The player with the least ranking is:
The Player Name is Osman, age is 36 and has Rank 585
Do you want to remove this player (y/n)? y
The players in the clan are:
Abbas 20 343 Middle East
Ali 24 454 North America
Ahmed 40 345 South America
```

Sample run:

```
Enter Player id (-1 to exit), age, rank, and server:
Abbas 20 343 Middle East
Enter Player id (-1 to exit), age, rank, and server:
Ali 24 454 North America
Enter Player id (-1 to exit), age, rank, and server:
Ahmed 40 345 South America
Enter Player id (-1 to exit), age, rank, and server:
Osman 36 585 South America
Enter Player id (-1 to exit), age, rank, and server:
-1
Enter a Server: Asia
No one from this server participated
The player with the least rank is:
The Player Name is Osman, age is 36 and has Rank 585
Do you want to remove this player (y/n)? n
The players in the clan are:
Abbas 20 343 Middle East
Ali 24 454 North America
Ahmed 40 345 South America
Osman 36 585 South America
```

Write a Java program that manages athletes in a marathon. Here are the UML diagrams:

Athlete	3.5 points
- id: int - age: int - distance: double - nationality: String	1
+ Athlete() + Athlete(id: int, age: int, distance: double, nationality: String) + setters + getters + getCompletionPercentage(): double	0.5 0.5 0.5 0.5 0.5

Marathon	7.5 points
- athlete[]: Athlete - numAthletes: int + MAX_SIZE: public static final int	1
+ Marathon() + getNumAthletes(): int + addAthlete(a: Athlete): void + removeAthlete(i: int): void + displayAthlete(i: int): void + getBestAthlete(nationality: String, fullMarathon: boolean): int + findLeastCompletionPercentage(): int	1 0.5 1 1 0.5 1 1.5

TestMarathon	4 points
+ main(): void	4

The Class: Athlete

As shown in the UML diagram, write the class Athlete that has the attributes:

- **id**: the id of the athlete.
- **age**: the athlete's age.
- **distance**: the distance the athlete covered in the marathon.
- **nationality**: the athlete's nationality.

The methods of this class are:

- **Athlete()**: A default constructor that sets id, age, and distance to -1 and nationality to "NA".
- **Athlete**(id,age,distance,nationality): A constructor that initializes new athlete with the initial values from the user.
- **Setter methods** (one for each): That sets the values for: id, age, distance, and nationality.

- **Getter Methods** (one for each): That returns the values for: id, age, distance, and nationality.
- **getCompletionPercentage**: return the completion percentage. Use the following table to calculate how much did the athlete complete of the marathon.

Age restriction	Total distance
Age < 18	21.1 km
Age >= 18	42.2 km

The Class: Marathon

The class **Marathon** contains an array of objects **athletes[ ]** that contains all the athletes that will compete in the marathon. It also contains the variable **numAthletes** that represents the current number of athletes stored in the array of objects **athletes[ ]**.

Note that the maximum number of atheletes in the list is 100 (hint: **MAX_SIZE** constant).

The methods of this class are:

- **Marathon()**: a constructor that initializes the attributes and creates an array of athletes of size 100
- **getNumAthletes**: returns the current number of athletes stored in the array.
- **addAthlete**: this method will add an athlete. If it is not possible to add the athlete, you should print an error message "**ERROR Adding**".
- **removeAthlete**: remove the athlete with index 'i' and put the last athlete in its place. Otherwise print "**ERROR Removing**".
- **displayAthlete**: displays the details for the athlete with index 'i' in the array, otherwise print "**ERROR Printing**". It should display the athlete information as follows:

```
The Saudi athlete no. 1, age 20 has covered 15 Km (%35.54502369668246).
```

- **getBestAthlete**: returns the index of the athlete with the best completion percentage from a given "nationality". If the given boolean is false, only compare athletes younger than 18. Otherwise, compare athletes that are 18 or older. If no such athlete exists, then return -1.
- **findLeastCompletionPercentage**: return the index of the athlete with the least completion percentage. If more than one has the same percentage, return the index of the first one. If no athletes exist, return -1.

The Main Class: TestMarathon

- **main**: the main method does the following:
  1. Create a **Marathon** object.
  2. (1 point) Ask the user to enter athletes' information. Keep reading and storing athletes' information in array athletes[] until user enters (-1) as the athlete's id.
  3. (0.5 point) Ask the user to enter a nationality, then print their best athlete in the full marathon and the half marathon. If no athlete with the given nationality exist, then print "No one from this nationality participated."
  4. (1 point) Display the athlete with the least completion percentage.
  5. Ask the user if they want to remove him/her from the marathon:
     a. (0.5 points) If "yes", remove the athlete from the Marathon object.
  6. (1 point) Finally, display all Athletes.

Sample run:

```
Enter athlete id (-1 to exit), age, distance, and nationality:
-1
```

Sample run:

```
Enter athlete id (-1 to exit), age, distance, and nationality:
111 20 30 Saudi
Enter athlete id (-1 to exit), age, distance, and nationality:
-1
Enter a nationality: Saudi
The best Saudi in the marathon is
The Saudi athlete no. 111, age 20 has covered 30.0 Km (%71.09004739336491).
The athlete with the least completion percentage is:
The Saudi athlete no. 111, age 20 has covered 30.0 Km (%71.09004739336491).
Do you want to remove this athlete (y/n)? y
This marathon has no athletes.
```

Sample run:

```
Enter athlete id (-1 to exit), age, distance, and nationality:
111 20 30 Saudi
Enter athlete id (-1 to exit), age, distance, and nationality:
222 24 28 USA
Enter athlete id (-1 to exit), age, distance, and nationality:
333 13 11 Saudi
Enter athlete id (-1 to exit), age, distance, and nationality:
444 33 39 Saudi
Enter athlete id (-1 to exit), age, distance, and nationality:
-1
Enter a nationality: Saudi
The best Saudi in the marathon is
The Saudi athlete no. 444, age 33 has covered 39.0 Km (%92.4170616113744).
The athlete with the least completion percentage is:
The Saudi athlete no. 333, age 13 has covered 11.0 Km (%52.132701421800945).
Do you want to remove this athlete (y/n)? y
The athletes are:
The Saudi athlete no. 111, age 20 has covered 30.0 Km (%71.09004739336491).
The USA athlete no. 222, age 24 has covered 28.0 Km (%66.35071090047393).
The Saudi athlete no. 444, age 33 has covered 39.0 Km (%92.4170616113744).
```

Sample run:

```
Enter athlete id (-1 to exit), age, distance, and nationality:
111 20 30 Saudi
Enter athlete id (-1 to exit), age, distance, and nationality:
222 24 28 USA
Enter athlete id (-1 to exit), age, distance, and nationality:
333 40 11 Saudi
Enter athlete id (-1 to exit), age, distance, and nationality:
444 33 39 Saudi
Enter athlete id (-1 to exit), age, distance, and nationality:
-1
Enter a nationality: UAE
No one from this nationality participated
The athlete with the least completion percentage is:
The Saudi athlete no. 333, age 13 has covered 11.0 Km (%52.132701421800945).
Do you want to remove this athlete (y/n)? n
The athletes are:
The Saudi athlete no. 111, age 20 has covered 30.0 Km (%71.09004739336491).
The USA athlete no. 222, age 24 has covered 28.0 Km (%66.35071090047393).
The Saudi athlete no. 333, age 13 has covered 11.0 Km (%52.132701421800945).
The Saudi athlete no. 444, age 33 has covered 39.0 Km (%92.4170616113744).
```