Project Information

Subject:

Games Company System.

Presented by: **Group:** 4

Lama Bugis	Batool Moneer	Aseel Allehyani
442018897	442019843	442010835

Supervised by:

Dr. Seereen Noorwali

The Project's Tasks:

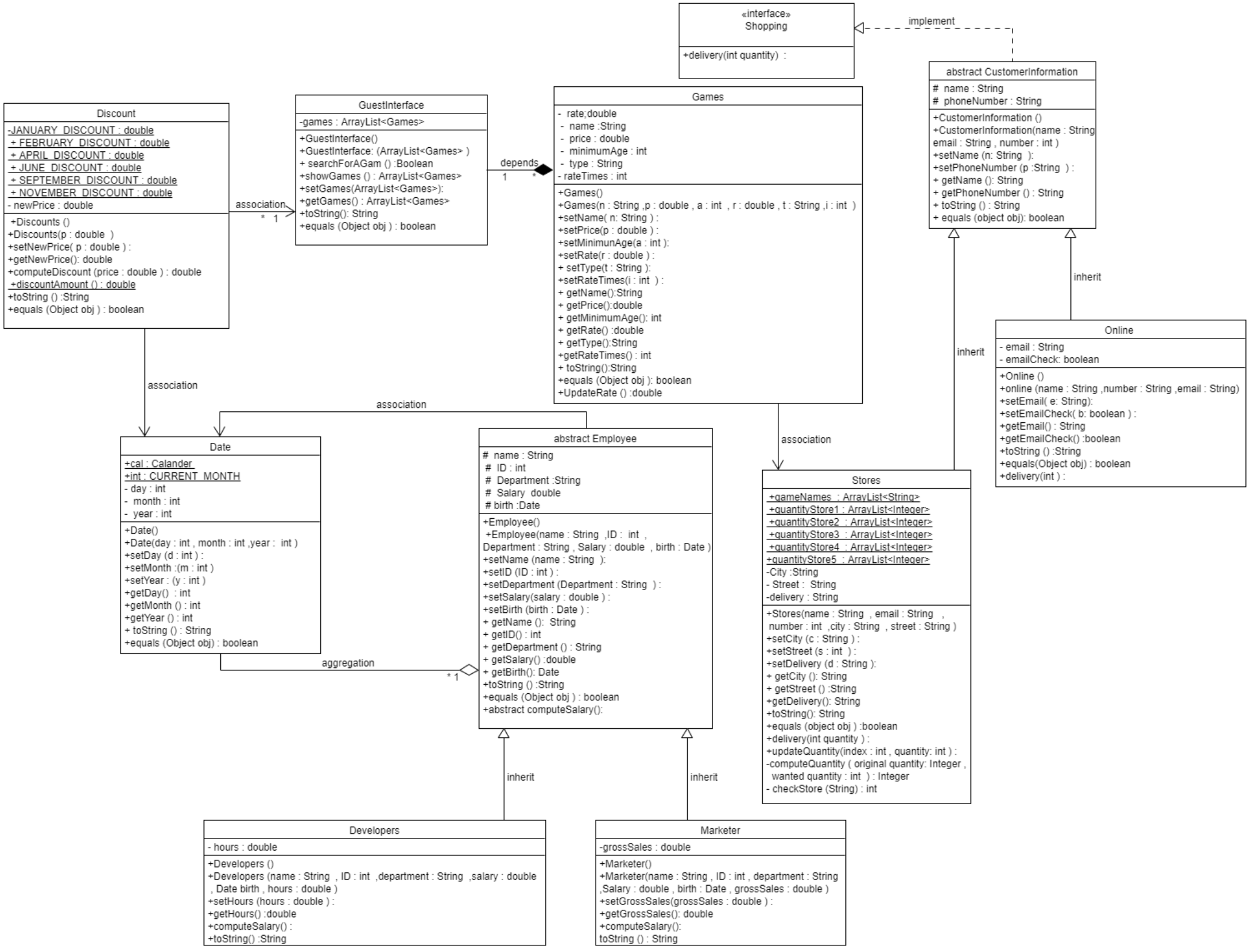
Our project is going to be for both the administration and customers, and there will be specific tasks for each of them.

The tasks for our customers:

- Display all games with its information.
- Search for a specific game.
- Display our stores.
- Allow the customer to rate any game.
- Make a discount in specific months

The administration's tasks:

- Display all employees' information.
- Make a salary bonus for some employees.
- Add a game.
- Display monthly sales for the games.
- Display all games with its information.


«interface»
Shopping
+delivery(int quantity) :
implement
abstract CustomerInformation
name : String
phoneNumber : String
+CustomerInformation ()
+CustomerInformation(name : String
email : String , number : int)
+setName (n: String):
+setPhoneNumber (p :String) :
+ getName (): String
+ getPhoneNumber () : String
+ toString () : String
+ equals (object obj): boolean
Discount
-JANUARY_DISCOUNT : double
+ FEBRUARY_DISCOUNT : double
+ APRIL_DISCOUNT : double
+ JUNE_DISCOUNT : double
+ SEPTEMBER_DISCOUNT : double
+ NOVEMBER_DISCOUNT : double
- newPrice : double
+Discounts ()
+Discounts(p : double)
+setNewPrice(p : double) :
+getNewPrice(): double
+computeDiscount (price : double) : double
+discountAmount () : double
+toString () :String
+equals (Object obj) : boolean
association
* 1
GuestInterface
-games : ArrayList<Games>
+GuestInterface()
+GuestInterface: (ArrayList<Games>)
+ searchForAGam () :Boolean
+showGames () : ArrayList<Games>
+setGames(ArrayList<Games>):
+getGames() : ArrayList<Games>
+toString(): String
+equals (Object obj) : boolean
depends
1 *
Games
- rate;double
- name :String
- price : double
- minimumAge : int
- type : String
- rateTimes : int
+Games()
+Games(n : String ,p : double , a : int , r : double , t : String ,i : int)
+setName(n: String) :
+setPrice(p : double) :
+setMinimunAge(a : int):
+setRate(r : double) :
+ setType(t : String):
+setRateTimes(i : int) :
+ getName():String
+ getPrice():double
+ getMinimumAge(): int
+ getRate() :double
+ getType():String
+getRateTimes() : int
+ toString():String
+equals (Object obj): boolean
+UpdateRate () :double
inherit
Online
- email : String
- emailCheck: boolean
+Online ()
+online (name : String ,number : String ,email : String)
+setEmail(e: String):
+setEmailCheck(b: boolean) :
+getEmail() : String
+getEmailCheck() :boolean
+toString () :String
+equals(Object obj) : boolean
+delivery(int) :
inherit
association
association
association
Date
+cal : Calander
+int : CURRENT_MONTH
- day : int
- month : int
- year : int
+Date()
+Date(day : int , month : int ,year : int)
+setDay (d : int) :
+setMonth :(m : int)
+setYear : (y : int)
+getDay() : int
+getMonth () : int
+getYear () : int
+ toString () : String
+equals (Object obj) : boolean
aggregation
* 1
abstract Employee
name : String
ID : int
Department :String
Salary double
birth :Date
+Employee()
+Employee(name : String ,ID : int ,
Department : String , Salary : double , birth : Date)
+setName (name : String):
+setID (ID : int) :
+setDepartment (Department : String) :
+setSalary(salary : double) :
+setBirth (birth : Date) :
+ getName (): String
+ getID() : int
+ getDepartment () : String
+ getSalary() :double
+ getBirth(): Date
+toString () :String
+equals (Object obj) : boolean
+abstract computeSalary():
Stores
+gameNames : ArrayList<String>
+quantityStore1 : ArrayList<Integer>
+quantityStore2 : ArrayList<Integer>
+quantityStore3 : ArrayList<Integer>
+quantityStore4 : ArrayList<Integer>
+quantityStore5 : ArrayList<Integer>
-City :String
- Street : String
-delivery : String
+Stores(name : String , email : String ,
number : int ,city : String , street : String)
+setCity (c : String) :
+setStreet (s : int) :
+setDelivery (d : String):
+ getCity (): String
+ getStreet () :String
+getDelivery(): String
+toString(): String
+equals (object obj) :boolean
+delivery(int quantity) :
+updateQuantity(index : int , quantity: int) :
-computeQuantity (original quantity: Integer ,
wanted quantity : int) : Integer
- checkStore (String) : int
inherit
inherit
Developers
- hours : double
+Developers ()
+Developers (name : String , ID : int ,department : String ,salary : double
, Date birth , hours : double)
+setHours (hours : double) :
+getHours() :double
+computeSalary() :
+toString() :String
Marketer
-grossSales : double
+Marketer()
+Marketer(name : String , ID : int , department : String
,Salary : double , birth : Date , grossSales : double)
+setGrossSales(grossSales : double) :
+getGrossSales(): double
+computeSalary():
toString () : String

UML Description

Our project contains 1 interface and 10 classes. 1 of these classes is abstract, 4 are subclasses, and the rest are concrete classes.

The details of the interface and classes are shown below:

Interfaces:

1- **Shopping:**
- Has a single method named (delivery).

Classes:

1. **CustomerInformation:**
   - Abstract class, implements the interface (Shopping).
   - Has 2 attributes that stores the information of each customer (name and phoneNumber).
   - Has the special methods (constructors – setters and getters – toString – equals).

2. **Stores:**
   - Subclass of CustomerInformation.
   - Has a 2D array (stores) to store the name and location of our stores, and 6 ArrayLists as class fields. 1 to store the games' names and the others are for storing the quantity in each store. Their names are (gameNames, quantityStore1, quantityStore2, quantityStore3, quantityStore4, quantityStore5).
   - Has (city, street, and delivery) which are the information of shipping to the customer as attributes.
   - Has the special methods (constructors – setters and getters – toString – equals), a body of (delivery) method that was declared in the (Shopping) interface, the method (checkStore) to check if there is a store in the customer's city, a method to compute if the store has the quantity that the customer wants (computeQuantity), and a method to update the quantity after the customer buys the game (updateQuantity).

3. **Games:**
   - Uses (Stores)'s class fields.
   - Has the information of the game as attributes (name, rate, price, minimumAge, type, and rateTimes).
   - Has the special methods (constructors – setters and getters – toString – equals) and the method (updateRate) to update the rate of a game after the customer rates it.

4. **GuestInterface:**
    - Depends on (Games).
    - Uses a static method from (Discounts).
    - Has an attribute (games ArrayList) of type (Games).
    - Has the special methods (constructors – setters and getters – toString – equals) and the method (searchForAGame) for the customer if he wants to check if a game exists or not.

5. **Discount**
    - Has a final amount of discounts for some months as class fields (JANUARY_DISCOUNT, FEBRUARY_DISCOUNT, APRIL_DISCOUNT, JUNE_DISCOUNT, SEPTEMBER_DISCOUNT, NOVEMBER_DISCOUNT).
    - Has an attribute (newPrice) to store the game's new price after the discount.
    - Has (constructors and setters/getters), the method (computeDiscount) to calculate the new price after discount. It chooses the discount based on the current month that is known by (Date), a static method (discountAmount) to return the amount of discount, and the overridden methods (toString and equals).

6. **Date:**
    - Has (day, month, year) as attributes.
    - Has a static final field (CURRENT_MONTH) that stores the current month and (cal) as a static field of type (Calender).
    - Has the special methods (constructors – setters and getters – toString – equals).

7. **Employee:**
    - Has the employee's information as attributes (name, ID, department, salary, birth "of type (Date)").
    - Has the special methods (constructors – setters and getters – toString – equals), and an abstract method (computeSalary) to update the salary of marketing and develovers employees based on their work.

8. **Marketetr:**
    - Subclass of (Employee).
    - Has the special methods (constructors – setters and getters – toString – equals) and the body of the abstract method (computeSalary) from the superclass

9. **Developers:**
    - Subclass of (Employee).
    - Has the special methods (constructors – setters and getters – toString – equals) and the body of the abstract method (computeSalary) from the superclass.

10. **Online:**
    - Subclass of (CustomerInformation).
    - Has the attributes (email and emailCheck) to store the email of the client and if the amail is valid or not respectively.
    - Has the special methods (constructors – setters and getters – toString – equals) and the body of the abstract method (delivey) from the superclass