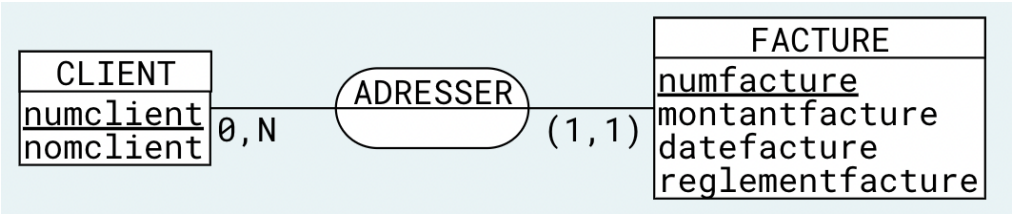
# Documentation des Classes Client et Facture

**Schéma du projet :**




## Classe Client

La classe Client représente un client dans un système de gestion de factures. Elle gère les informations du client (nom), ainsi que la création et le calcul des factures associées.
Attributs :

- nom : Le nom du client.
- factureList : Liste des factures du client.
- factRegle : Liste des factures réglées du client.
- nbClient : Liste globale de tous les clients.

```
private String nom;

private List<Facture> factureList = new ArrayList<Facture>();

private List<Facture> factRegle = new ArrayList<Facture>();

private static List<Client> nbClient = new ArrayList<Client>();
```

Méthodes :

- Client(String nom) :
  - Description : Constructeur qui permet de créer un client en fournissant un nom. Le client est ensuite ajouté à la liste globale des clients.
  - Paramètre : nom (String) : Le nom du client.
- getNom() :
  - Description : Retourne le nom du client.
  - Retour : String : Le nom du client.
- setNom(String nom) :
  - Description : Modifie le nom du client.
  - Paramètre : nom (String) : Le nouveau nom du client.
- createFacture(int montant) :
  - Description : Crée une facture pour le client avec un montant spécifié. Si le montant est invalide (inférieur ou égal à zéro), une exception Erreur est levée.
  - Paramètre : montant (int) : Le montant de la facture.
  - Retour : Facture : L'objet Facture créé.

```java
    public Client(String nom)
    {
            this.nom = nom;
            nbClient.add(this);
    }

    public void setfactureList(Facture t){
            factureList.remove(t);
    }

    public void addListFact(Facture t){
    factureList.add(t);
}

    /**
     * Retourne le nom du client.
     * @return le nom du client.
     */

    public String getNom()
    {
            return this.nom;
    }

    /**
     * Modifie le nom du client.
     * @param nom le nom du client.
     */

    public void setNom(String nom)
    {
            this.nom = nom;
    }

    /**
     * Créé une facture.
     * @param montant Le montant de la facture.
     * @return la facture créée.
     */

    public Facture createFacture(int montant)
    throws IllegalArgumentException
    {

                    Facture facture = new Facture(this , montant);
                    if(montant <= 0) {throw new IllegalArgumentException("Le montant d'une facture ne peut pas être négatif.");

                    }
                    factureList.add(facture);
                    return facture ;



    }
```

- createFacture(int montant, boolean reglee) :
    - Description : Crée une facture pour le client avec un montant et un statut de règlement spécifié (true si réglée, false sinon). Si la facture est réglée, elle est ajoutée à la liste des factures réglées, sinon elle est ajoutée aux factures non réglées.
    - Paramètre :
        - montant (int) : Le montant de la facture.
        - reglee (boolean) : Le statut de règlement de la facture.
    - Retour : Facture : L'objet Facture créé.
    - Exception : Lève une exception Erreur si le montant est inférieur ou égal à zéro.
- sommeMontants() :
    - Description : Calcule la somme totale des montants des factures non réglées du client.
    - Retour : int : La somme des montants des factures non réglées.
- getFactures() :
    - Description : Retourne une copie de la liste des factures non réglées du client.
    - Retour : List<Facture> : Une liste des factures non réglées du client.
- facturesReglees() :
    - Description : Retourne une copie de la liste des factures réglées du client.
    - Retour : List<Facture> : Une liste des factures réglées du client.

```java
public int sommeMontants()
{
        int s = 0;
        for(Facture fact : factureList){
                s += fact.getMontant();
        }
        return s;
}
/**
 * Retourne une copie de la liste des factures du client.
 * @return une copie de la liste des factures du client.
 */

 public List<Facture> getFactures()
 {
         List<Facture> test = new ArrayList<Facture>();
         for(Facture fact : factureList){
                 test.add(fact);
         }
         return test;
 }

/**
 * Créé une facture en précisant si elle est reglée.
 * @param montant Le montant de la facture.
 * @param reglée Vrai si la facture est reglée.
 * @return la facture créée.
 */

public Facture createFacture(int montant, boolean reglee)
throws IllegalArgumentException{
        Facture factReglee = new Facture(this , montant, reglee);
        if(montant <= 0) throw new IllegalArgumentException("Le montant d'une facture ne peut pas être négatif.");
        if(reglee == true){
                factRegle.add(factReglee);
        }
        return factReglee;


}

/**
 * Retourne la liste des factures reglées.
 * @return la liste des factures reglées.
 */

public List<Facture> facturesReglees()
{
        List<Facture> t = new ArrayList<Facture>();
        for(Facture z : factRegle){
                t.add(z);
        }
        return t;
}
```

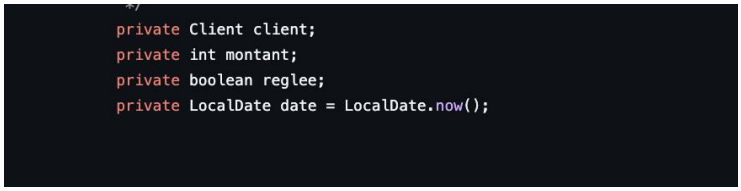
- tous() :
  - Description : Retourne une copie de la liste de tous les clients créés.
  - Retour : List<Client> : Liste de tous les clients existants.
- delete() :
  - Description : Supprime le client de Collections des clients.

```java
public static List<Client> tous()
{
        List<Client> t = new ArrayList<Client>();
        for(Client z : nbClient){
                t.add(z);
        }
        return t;
}

/**
 * Supprime le client.
 */

public void delete()
{
        nbClient.remove(this);
}
```

# Classe Facture

La classe Facture représente une facture associée à un client et contient les informations liées à la facture telles que le montant, le statut de règlement, et la date de création.
Attributs :

- client : Le client auquel la facture est associée.
- montant : Le montant de la facture.
- reglee : Le statut de la facture (réglée ou non).
- date : La date de création de la facture.



```
*/
private Client client;
private int montant;
private boolean reglee;
private LocalDate date = LocalDate.now();
```

Méthodes :

- Facture(Client client, int montant, boolean reglee) :
  - Description : Constructeur qui permet de créer une facture en spécifiant le client, le montant et le statut de la facture (réglée ou non).
  - Paramètre :
    - client (Client) : Le client auquel la facture est associée.
    - montant (int) : Le montant de la facture.
    - reglee (boolean) : Le statut de la facture (true si réglée, false sinon).
- Facture(Client client, int montant) :
  - Description : Constructeur qui permet de créer une facture en spécifiant le client et le montant. Le statut de la facture est par défaut non réglée (false).
  - Paramètre :
    - client (Client) : Le client auquel la facture est associée.
    - montant (int) : Le montant de la facture.
- getClient() :
  - Description : Retourne le client auquel la facture est associée.
  - Retour : Client : Le client associé à la facture.

- getMontant() :
  - Description : Retourne le montant de la facture.
  - Retour : int : Le montant de la facture.

```java
public Facture(Client client , int montant , boolean reglee){
        if (montant > 0) this.montant = montant ;
        this.reglee = reglee;
        this.client =  client;
}


public Facture(Client client , int montant){
        if (montant > 0 ) this.montant = montant ;
        this.client = client;
}

public Client getClient()
{
        return client;
}

/**
 * Retourne le montant de la facture.
 * @return le montant de la facture.
 */

public int getMontant()
{
        return this.montant;
}
```

- estReglee() :
  - Description : Retourne true si la facture est réglée, sinon retourne false.
  - Retour : boolean : Le statut de la facture (réglée ou non).
- getDate() :
  - Description : Retourne la date de création de la facture.
  - Retour : LocalDate : La date de création de la facture.
- delete() :
  - Description : Supprime la facture de la collections des factures non réglées du client.
- copie() :
  - Description : Crée une copie de la facture et l'ajoute à la liste des factures du client. La copie a les mêmes attributs que l'originale.
  - Retour : Facture : La copie de la facture créée.

```java
public boolean estReglee()
{
        if (reglee == true) return true;
        else return false;

}

/**
 * Retourne la date de la facture.
 * @return la date de la facture.
 */

public LocalDate getDate()
{
        return this.date;
}

/**
 * Supprime la facture
 */

public void delete()
{
        this.client.setfactureList(this);
}

/**
 * Duplique la facture.
 * @return une copie de la facture.
 */

public Facture copie()
{
        Facture t = new Facture(client , this.montant,this.reglee);
        this.client.addListFact(t);
        return t;

}
```

# Exception Erreur

L'exception Erreur est une classe personnalisée qui est levée lorsqu'un montant de facture est invalide (inférieur ou égal à zéro). Elle est utilisée pour garantir que les factures ne peuvent être créées qu'avec des montants valides.
Méthodes :

- getMessage() :
  - Description : Retourne un message d'erreur détaillant la cause de l'exception.
  - Retour : String : Le message d'erreur, qui dans ce cas précise que "Le montant d'une facture ne peut pas être négatif."

```
class Erreur extends Exception{

        public String getMessage(){

                return "Le montant d'une facture ne peut pas être négatif.";
        }

}
```

---

Résumé

- Classe Client : Permet de gérer les informations d'un client et la création, gestion, et calcul des factures. Elle inclut également la gestion des erreurs via l'exception Erreur qui est levée lorsque des montants de factures invalides sont spécifiés.
- Classe Facture : Représente une facture, permettant de gérer le client, le montant, le statut de règlement et la date de création de la facture. Elle permet aussi de supprimer ou de dupliquer une facture.
- Exception Erreur : Permet de signaler les erreurs liées à la création de factures avec des montants non valides.

---

Ainsi, avec ces deux classes, vous pouvez gérer une application de facturation où chaque client peut avoir des factures réglées ou non, avec la possibilité de calculer des montants totaux, supprimer des factures et gérer les erreurs de manière adéquate.