Conception programmation objet

Enchère

Daryl Martin-Dipp, Maxime Wang ALTERNANCE

Tables des matières

Première Itération :

Diagramme de cas d'utilisation

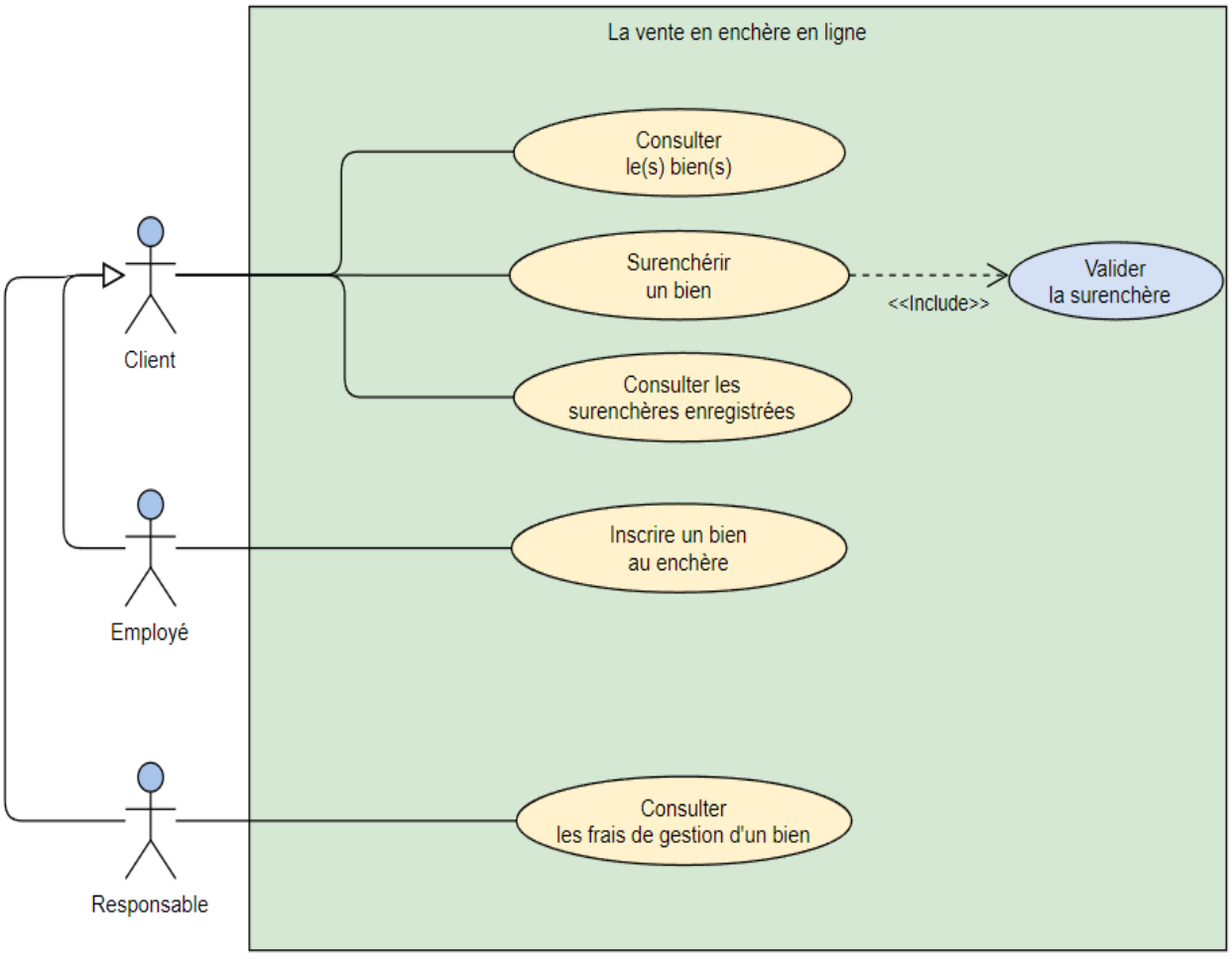
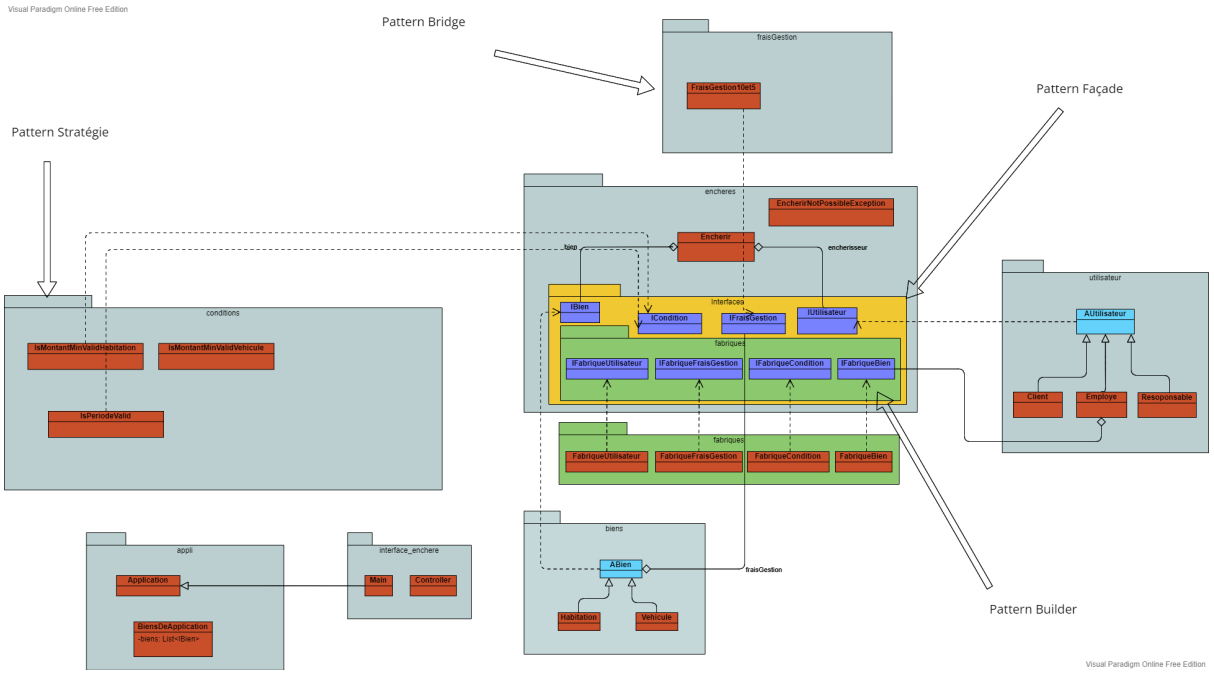


Diagramme de classes



Design Pattern Utilisés :

Pattern de création :

Problème :

Imaginons que la première version de notre application ne propose que l'enchère d'habitation, du coup la majeure partie de notre code est donc située dans la classe `Habitation`.

Au bout d'un certain temps, notre application devient populaire et on nous demande d'ajouter des biens à vendre. La majeure partie est actuellement couplée à la classe `Habitation`. Pour pouvoir ajouter des `Véhicule` dans l'application, il faudrait revoir la base du code. De plus, si on décide plus tard d'ajouter un autre type de bien dans l'application, il faudra effectuer à nouveau ces changements.

Par conséquent, on va se retrouver avec du code pas très propre, rempli de conditions qui modifient le comportement du programme en fonction de la classe des objets de bien.

Solution : Fabrique

Utilisation du patron de conception Fabrique va nous permettre de remplacer les appels directs au constructeur de l'objet (à l'aide de l'opérateur `new`) en appelant une méthode *fabrique* spéciale au bien, si l'on reste dans le contexte de l'exemple du dessus.
On va donc avoir une interface pour créer des objets dans une classe mère, mais on délègue le choix des types d'objets à créer aux sous-classes. Ainsi dans notre application, nous avons créé des fabriques pour les biens, les utilisateurs, les conditions et les frais de gestion .

Pattern de structure :

Problème :

Il existe une dépendance du type "Bien" stable du code de consultation des frais de gestion qui est elle instable.
– SRP : la classe contient des bouts de codes n'évoluant pas au même rythme
– OCP : modification de la classe en cas de changement des frais de gestion

Solution : Bridge

Utilisation du patron de conception bridge qui résout ce problème en utilisant la composition à la place de l'héritage. Pour ce faire, on insère une des dimensions dans une hiérarchie de classes séparée ici "IFraisGestion" afin que la classe originale puisse référencer un objet de cette nouvelle hiérarchie, plutôt que de réunir tous les états et comportements à l'intérieur d'une même classe. Ainsi nous permettons de séparer une grosse classe ou un ensemble de classes connexes en deux hiérarchies "abstraction et implémentation" qui peuvent évoluer indépendamment l'une de l'autre. Ainsi dans notre application, nous avons créé l'interface et la classe abstract pour gérer les frais de gestion.

Problème :
Pour notre application si nous avons maintenant envie de modifier en temps réel les conditions pour qu'une surenchère soit possible. On entend que ça n'a pas l'air de quelque chose de facile.

Solution : Stratégie
Utilisation du patron de conception stratégie qui consiste à définir plusieurs implémentations interchangeables pour une même méthode, changer le comportement d'un objet pendant l'exécution et définir une famille d'algorithmes pour une même méthode et pouvoir en choisir un et le remplacer pendant l'exécution.

Problème :
Imaginez que nous souhaitons adapter notre code pour manipuler un ensemble d'objets qui appartiennent à une librairie comme celle du biens dans notre cas. Normalement, on initialise tous ces objets en premier, gardez la trace des dépendances et appelez les méthodes dans le bon ordre, etc.

Par conséquent, la logique métier de nos classes devient fortement couplée avec les détails de l'implémentation des classes externes, rendant cette logique difficile à comprendre et à maintenir.

Solution : Façade
Utilisation du patron de conception façade, étant une classe qui procure une interface offrant un accès simplifié à une librairie, à n'importe quel ensemble complexe. Ainsi dans notre application, nous avons créé des interfaces pour différentes classes comme l'interface pour les biens, les frais de gestion, les conditions et les utilisateurs.

Problème :
Pour notre application si nous avons beaucoup de code similaire, sachant qu'on aura des processus décisionnels redondants dans les deux sous-classes. Par exemple, ajouter de nouvelles classes (comme une 3ème condition) nécessitera de recoder la procédure de décision dans chacune de ces nouvelles sous-classes. Dans ce cas, chaque changement de procédure est multiplié par le nombre de sous-classes.

Solution : Template Méthode

Utilisation du patron de conception template méthode qui consiste à factoriser la procédure de décision dans la classe mère ACondtionMax puis coder la procédure sous forme d'une suite d'opérations abstraites qui seront implémentées par les sous-classes.

Dépôts code :

Package Appli

Classe Application

```
package appli;

import biens.Habitation;
import conditions.IsMontantMinValidVehicule;
import conditions.IsPeriodeValid;
import encheres.interfaces.IBien;
import encheres.interfaces.fabriques.IFabriqueCondition;
import fabriques.FabriqueCondition;
import fraisGestion.FraisGestion10et5;

import java.util.Calendar;
import java.util.GregorianCalendar;
import java.util.List;

/**
* Modélise l'application lancant une session d'enchere.
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public class Application {

    /**
     * Creation et lancement de l'enchere
     */
    public static void main(String[] args) {
        // Dates
        Calendar dateD = new GregorianCalendar(2021, Calendar.OCTOBER, 1);
        Calendar dateF = new GregorianCalendar(2022, Calendar.DECEMBER, 1);

        IFabriqueCondition fabriqueCondition = new FabriqueCondition();
        IBien habitation = new Habitation(
                "immeuble",
```

```
                200000,
                dateD,
                dateF,
                new FraisGestion10et5(),
                "Paris",
                5
                );

        habitation.setFraisGestion(new FraisGestion10et5());

        BiensDeApplication.ajouterBien(habitation);
        fabriqueCondition.fabriqueCondition("habitation", new
IsMontantMinValidVehicule(),
                new IsPeriodeValid());
    }

    public static List<IBien> getBiens() {
        return BiensDeApplication.getBiens();
    }

}
```

Classe BiensDeApplication

```
package appli;

import encheres.interfaces.IBien;

import java.util.ArrayList;
import java.util.List;

/**
* Modélise la liste des biens de notre application d'enchere.
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public class BiensDeApplication {

    /**
     * La liste de biens mis en vente a notre enchere
     */
    private static List<IBien> biens = new ArrayList<>();

    /**
     * Permet de connaître la liste de biens mis en vente
     * @return la liste de biens
     */
```

```java
    public static List<IBien> getBiens() {
        return biens;
    }

    /**
     * Permet d'ajouter un bien dans la liste de biens mis en vente
     * @param bien le bien a ajoute a la liste de biens mis en vente
     */
    public static void ajouterBien(IBien bien) {
        biens.add(bien);
    }

    /**
     * Permet de suppprimer de la liste tous les biens qui s'y trouve
     */
    public static void clearListBiens() {
        biens.clear();
    }

}
```

Package biens

Classe ABien

```java
package biens;

import encheres.interfaces.IBien;
import encheres.interfaces.ICondition;
import encheres.interfaces.IFraisGestion;
import encheres.interfaces.IUtilisateur;
import encheres.Enchere;
import encheres.EncherirNotPossibleException;

import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.HashMap;
import java.util.List;

/**
* Modélise un bien en general.
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public abstract class ABien implements IBien {

    /**
     * Description du bien
     */
    private String description;

    /**
     * Montant de depart du bien
     */
    private double montantD;

    /**
     * Montant actuel du bien
     */
    private double montant;

    /**
     * Date de début de l'enchere du bien
     */
    private Calendar dateD;

    /**
     * Date de fin de l'enchere du bien
     */
    private Calendar dateF;

    /**
     * Frais de gestion du bien
     */
    private IFraisGestion fraisGestion;

    /**
     * Liste representant les surencheres enregistrees d'un bien
     */
    private HashMap<IUtilisateur, Double> surenchèresEnregistrees;
```

```java
    /**
     * La liste correspondant aux conditions d'un bien
     */
    private static List<ICondition> conditions;

    /**
     * Constructeur du bien
     * @param description la description du bien
     * @param montantD le montant depart du bien
     * @param dateD la date de debut de l'enchere du bien
     * @param dateF la date de fin de l'enchere du bien
     * @param fraisGestion les frais de gestion du bien
     */
    public ABien(String description, double montantD, Calendar dateD, Calendar
dateF, IFraisGestion fraisGestion) {
        this.description = description;
        this.montantD = montantD;
        this.montant = montantD;
        this.dateD = dateD;
        this.dateF = dateF;
        this.fraisGestion = fraisGestion;
        this.surencheresEnregistrees = new HashMap<>();
    }


    // Getters

    /**
     * Permet de connaître les conditions qu'un bien doit valider pour etre
surrencherit
     * @return la liste des conditions qu'un bien doit valider
     */
    public static List<ICondition> getConditions() {
        return conditions;
    }

    @Override
    public String getDescription() {
        return description;
    }

    @Override
    public double getMontant() {
        return montant;
    }

    @Override
    public IBien getBien() {
        return this;
    }

    @Override
    public double getMontantD() {
        return montantD;
    }

    @Override
```

```
    public Date getDateD() {
        return dateD.getTime();
    }

    @Override
    public Date getDateF() {
        return dateF.getTime();
    }

    @Override
    public HashMap<IUtilisateur, Double> getSurencheresEnregistrees() {
        return surencheresEnregistrees;
    }


    // Setters


    /**
     * Permet de mettre a jour les conditions qu'un bien doit faire valider avant
d'etre encherit
     * @param conditions la liste des conditions que devra suivre le bien pour
etre validé
     */
    public static void setConditions(List<ICondition> conditions) {
        ABien.conditions = conditions;
    }

    @Override
    public void setMontant(double montant) {
        this.montant = montant;
    }

    @Override
    public void setFraisGestion(IFraisGestion fraisGestion) {
        this.fraisGestion = fraisGestion;
    }

    @Override
    public void setSurencheresEnregistrees(IUtilisateur utilisateur, double
montant) {
        this.surencheresEnregistrees.put(utilisateur, montant);
    }

    @Override
    public String toString() {
        SimpleDateFormat dateFormatD = new SimpleDateFormat("dd-MMM-yyyy");
        SimpleDateFormat dateFormatF = new SimpleDateFormat("dd-MMM-yyyy");
        dateFormatD.setCalendar(dateD);
        String dateFormattedD = dateFormatD.format(dateD.getTime());

        dateFormatF.setCalendar(dateF);
        String dateFormattedF = dateFormatF.format(dateF.getTime());


        return "Bien : " + description +
                "\nmontant de depart : " + montantD +
```

```
                "\nmontant actuelle : " + montant +
                "\ndate de debut : " + dateFormattedD +
                "\ndate de fin : " + dateFormattedF ;
    }


// Methods

    @Override
    public void encherir(double montant) throws EncherirNotPossibleException {
        new Enchere(this, montant);
    }

    @Override
    public double consulterFraisGestion() {
        return fraisGestion.fraisGestionActuel(this);
    }
}
```

Classe Habitation

```
package biens;

import encheres.interfaces.IFraisGestion;

import java.util.Calendar;

/**
* Modélise une habitation.
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public class Habitation extends ABien {

    /**
     * La ville de l'habitation
     */
    private String ville;

    /**
     * Le nombre de piece de l'habitation
     */
    private int nbPiece;

    /**
     * Constructeur d'une habitation
     * @param description la description de l'habitation
     * @param montantD le montant depart de l'habitation
     * @param dateD la date de debut de l'habitation
     * @param dateF la date de fin de l'habitation
     * @param fraisGestion les frais de gestion de l'habitation
     * @param ville la ville de l'habitation
     * @param nbPiece le nombre de piece de l'habitation
     */
    public Habitation(String description, double montantD, Calendar dateD,
Calendar dateF, IFraisGestion fraisGestion,
                      String ville, int nbPiece) {
        super(description, montantD, dateD, dateF, fraisGestion);
        this.ville = ville;
        this.nbPiece = nbPiece;
    }

}
```

Classe Vehicule

```java
package biens;

import encheres.interfaces.IFraisGestion;

import java.util.Calendar;

/**
* Modélise un vehicule.
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public class Vehicule extends ABien {

    /**
     * La marque du vehicule
     */
    private String marque;

    /**
     * l'annee de fabrication du vehicule
     */
    private int anneeFabrication;

    /**
     * Constructeur du vehicule
     * @param description la description du vehicule
     * @param montantD le montant depart du vehicule
     * @param dateD la date de debut du vehicule
     * @param dateF la date de fin du vehicule
     * @param fraisGestion les frais gestion du vehicule
     * @param marque la marque du vehicule
     * @param anneeFabrication l'annee de fabrication du vehicule
     */
    public Vehicule(String description, double montantD, Calendar dateD, Calendar
dateF, IFraisGestion fraisGestion,
                    String marque, int anneeFabrication) {
        super(description, montantD, dateD, dateF, fraisGestion);
        this.marque = marque;
        this.anneeFabrication = anneeFabrication;
    }

}
```

Package conditions

Classe IsMontantMinValidHabitation

```java
package conditions;

import encheres.interfaces.IBien;
```

```java
import encheres.interfaces.ICondition;

/**
* Modélise une condition de l'enchere d'une habitation
* permettant la verification du mantant minimum pour un enrichissement
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public class IsMontantMinValidHabitation implements ICondition {

    @Override
    public boolean isConditionRespected(IBien bien, double montant) {
        if (bien.getSurencheresEnregistrees().isEmpty())
            return montant >= bien.getMontantD();
        else
            return montant > bien.getMontant() * 1.1;
    }

}
```

Classe IsMontantMinValidVehicule

```java
package conditions;

import encheres.interfaces.IBien;
import encheres.interfaces.ICondition;

/**
* Modélise une condition de l'enchere d'une habitation
* permettant la verification du mantant minimum pour un enrichissement
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public class IsMontantMinValidVehicule implements ICondition {

    @Override
    public boolean isConditionRespected(IBien bien, double montant) {
        if (bien.getSurencheresEnregistrees().isEmpty())
            return montant >= bien.getMontantD();
        else
            return montant > bien.getMontant();
    }

}
```

Classe IsPeriodValid

```java
package conditions;

import encheres.interfaces.IBien;
import encheres.interfaces.ICondition;

import java.util.Calendar;

/**
* Modélise une condition de l'enchere d'un bien
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public class IsPeriodeValid implements ICondition {

    @Override
    public boolean isConditionRespected(IBien bien, double montant) {
        return bien.getDateD().before(Calendar.getInstance().getTime()) &&
                bien.getDateF().after(Calendar.getInstance().getTime());
    }

}
```

Package encheres

Classe Enchere

```java
package encheres;

import biens.ABien;
import encheres.interfaces.IBien;
import encheres.interfaces.ICondition;

/**
* Modelise une enchere d'un bien.
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public class Enchere {

    /**
     * Represente le bien de l'enchere
```

```
     */
    private IBien bien;

    /**
     * Represente le montant de l'encherissement du bien
     */
    private double montant;

    /**
     * Constructeur de l'enchere à partir d'un bien et d'un montant
     * @param bien le bien que l'on souhaite encherir
     * @param montant le montant de l'encherissement
     * @throws EncherirNotPossibleException l'erreur rejete si l'encherissement
est impossible
     */
    public Enchere(IBien bien, double montant) throws EncherirNotPossibleException
{
        this.bien = bien;
        this.montant = montant;

        // commence la fonction de l'encherissement
        encherir();
    }

    /**
     * Permet de encherir le bien en verifiant si les conditions sont valides
     * @throws EncherirNotPossibleException
     */
    public void encherir() throws EncherirNotPossibleException {
        for (ICondition condi : ABien.getConditions())
            if (!condi.isConditionRespected(bien, montant)) {
                throw new EncherirNotPossibleException("L'enchère n'a pas pu
aboutir. Veuillez vérifier le montant et la date de l'enchère.");
            }

        bien.setMontant(montant);

    }
}
```

Classe EncherirNotPossibleException

```
package encheres;

/**
* L'erreur envoye lorsque l'encherissement d'un bien est impossible.
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public class EncherirNotPossibleException extends Exception{

    /**
     * Message d'erreur lorsque qu'un encherissement est impossible
```

```
     * @param errorMessage le message de l'erreur
     */
    public EncherirNotPossibleException(String errorMessage) {
        super(errorMessage);
    }
}
```

Package encheres.interfaces

Interface IBien

```
package encheres.interfaces;

import encheres.EncherirNotPossibleException;

import java.util.Date;
import java.util.HashMap;

/**
* L'interface d'un bien en general.
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public interface IBien {

    /**
     * Permet de connaître le bien
     * @return le bien
     */
    IBien getBien();

     /**
     * Permet de connaître la description du bien
     * @return le descriptif du bien
     */
    String getDescription();


    /**
     * Permet de connaître le montant de depart du bien
     * @return le montant de depart du bien
     */
    double getMontantD();

    /**
     * Permet de connaître le montant actuel bien
     * @return le montant actuel bien
     */
```

```
    double getMontant();

    /**
     * Permet de connaître la date de debut de l'enchere du bien
     * @return la date de debut de l'enchere du bien
     */
    Date getDateD();

    /**
     * Permet de connaître la date de fin de l'enchere du bien
     * @return la date de fin de l'enchere du bien
     */
    Date getDateF();

    /**
     * Permet de modifier le montant actuel du bien
     * @param montant le montant du bien
     */
    void setMontant(double montant);

    /**
     * Permet d'encherir un bien
     * @param montant le montant de l'encherissement
     * @throws EncherirNotPossibleException une erreur lorsque l'encherissement
est impossible
     */
    void encherir(double montant) throws EncherirNotPossibleException;

    /**
     * Permet de modifier les frais de gestion du bien
     * @param fraisGestion le frais de gestion du bien
     */
    void setFraisGestion(IFraisGestion fraisGestion);

    /**
     * Permet de consulter les frais gestion du bien
     * @return le montant des frais de gestion du bien
     */
    double consulterFraisGestion();

    /**
     * Permet d'enregistrer les surencherissements valides dans la liste prevu a
cette effet
     * @param utilisateur l'utilisateur qui a encheri sur le bien
     * @param montant le montant de son encherissement
     */
    void setSurencheresEnregistrees(IUtilisateur utilisateur, double montant);

    /**
     * Permet de connaître la liste des surencherissements enregistrees du bien
     * @return la liste des surencherissements du bien
     */
    HashMap<IUtilisateur, Double> getSurencheresEnregistrees();

}
```

Interface ICondition

```java
package encheres.interfaces;

/**
* L'interface d'une condition en general.
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public interface ICondition {

    /**
     * Permet de verifier si la condition est respectee
     * @return True si la condition est respectee, False dans le cas contraire
     */
    boolean isConditionRespected(IBien bien, double montant);

}
```

Interface IFraisGestion

```java
package encheres.interfaces;

/**
* L'interface representant les frais de gestion en general.
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public interface IFraisGestion {

    /**
     * Permet de connaître les frais de gestion actuel de ce bien
     * @param bien le bien ou l'on veut connaître les frais de gestion
     * @return le montant correspondant au frais de gestion de ce bien
     */
    double fraisGestionActuel(IBien bien);

}
```

Interface IUtilisateur

```java
package encheres.interfaces;

import encheres.EncherirNotPossibleException;

import java.util.HashMap;
import java.util.List;

/**
* L'interface d'un utilisateur en general.
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public interface IUtilisateur {

    /**
     * Permet de connaître le nom d'utilisateur
     * @return
     */
    String getNomUtilisateur();

    /**
     * Permet de connaître la liste de biens surencheri par cette utilisateur
     * @return la liste de biens surencheri par cette utilisateur
     */
    HashMap<IBien, Double> getBiensSurencheris();

    /**
     * Permet de connaître la liste de biens achetes par cette utilisateur
     * @return la liste de biens achetes par cette utilisateur
     */
    HashMap<IBien, Double> getBiensAchetes();

    /**
     * Permet de consultaer les biens mis en vente de l'application
     * @return la liste des biens mis en vente de l'application
     */
    List<IBien> consulterBiens();

    /**
     * Permet de surencherir un bien
     * @param bien le bien a surencherir
     * @param montant le montant du surencherissement
     * @throws EncherirNotPossibleException l'erreur rejete si l'encherissement
est impossible
     */
    void surencherir(IBien bien, double montant) throws
EncherirNotPossibleException;

    /**
     * Permet d'inscrire un bien dans l'application pour que les utilisateurs
puissent encherir dessus par la suite
     * @param type le type du bien
     * @param attributs les attributs du bien
     * @return le bien qui a ete demander d'etre inscrit
```

```
     */
    IBien inscrireBien(String type, Object... attributs);

    /**
     * Permet de consulter les frais de gestion d'un bien
     * @param bien le bien ou l'on veut consulter les frais de gestion
     * @return le montant des frais de gestion de ce bien
     */
    double consulterFraisGestion(IBien bien);

}
```

Package encheres.interfaces.fabriques

Interface IFabriqueBien

```
package encheres.interfaces.fabriques;

import encheres.interfaces.IBien;

/**
* L'interface d'une fabrique d'un bien.
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public interface IFabriqueBien {

    /**
     * Permet de fabriquer un bien
     * @param type le type de bien a fabrique
     * @param attributs les attributs de ce bien
     * @return le bien souhaitant etre fabrique
     */
    IBien fabriqueBien(String type, Object... attributs);

}
```

Interface IFabriqueCondition

```
package encheres.interfaces.fabriques;
```

```java
import encheres.interfaces.ICondition;

import java.util.List;

/**
* L'interface d'une fabrique de plusieurs conditions.
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public interface IFabriqueCondition {

    /**
     * Permet de fabriquer une condition
     * @param type le type de condition
     * @return la condition souhaitant etre fabrique
     */
    List<ICondition> fabriqueCondition(String type, ICondition... attributs);

}
```

Interface IFabriqueFraisGestion

```java
package encheres.interfaces.fabriques;

import encheres.interfaces.IFraisGestion;

/**
* L'interface d'une fabrique d'un frais de gestion.
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public interface IFabriqueFraisGestion {

    /**
     * Permet de fabriquer un frais de gestion
     * @param type le type de frais de gestion souhaitant etre fabrique
     * @return le frais de gestion souhaitant etre fabrique
     */
    IFraisGestion fabriqueFraisGestion(String type);

}
```

Interface IFabriqueUtilisateur

```java
package encheres.interfaces.fabriques;

import encheres.interfaces.IUtilisateur;

/**
* L'interface d'une fabrique d'un utilisateur.
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public interface IFabriqueUtilisateur {

    /**
     * Permet de fabriquer des utilisateurs
     * @param type le type d'utilisateur
     * @param nomUtilisateur le nom d'utilisateur
     * @return
     */
    IUtilisateur fabriqueUtilisateur(String type, String nomUtilisateur);

}
```

Package fabriques

Classe FabriqueBien

```java
package fabriques;

import biens.Habitation;
import biens.Vehicule;
import encheres.interfaces.IBien;
import encheres.interfaces.fabriques.IFabriqueBien;
import encheres.interfaces.IFraisGestion;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;
import java.util.Locale;

/**
* Modelise une fabrique d'un bien.
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public class FabriqueBien implements IFabriqueBien {

    @Override
```

```
    public IBien fabriqueBien(String type, Object... attributs) {

        if (attributs == null)
            throw new IllegalArgumentException("Les attributs ne sont pas définis
! ! !");

        type.toLowerCase(Locale.ROOT);

        switch (type) {

            case "habitation": {
                List<Object> list = getAttributs(attributs);
                return new Habitation((String) list.get(0), (double) list.get(1),
(Calendar) list.get(2),
                        (Calendar) list.get(3), (IFraisGestion) list.get(4),
(String) list.get(5), (int) list.get(6));
            }
            case "vehicule" : {
                List<Object> list = getAttributs(attributs);
                return new Vehicule((String) list.get(0), (double) list.get(1),
(Calendar) list.get(2),
                        (Calendar) list.get(3), (IFraisGestion) list.get(4),
(String) list.get(5), (int) list.get(6));
            }
            default:
                throw new IllegalArgumentException("Type n'est pas defini");
        }

    }

    /**
     * Permet d'extraire les donnees passees en parametre
     * @param attributs les attributs a extraire
     * @return une liste d'attribut
     */
    private List<Object> getAttributs(Object[] attributs) {
        List<Object> list = new ArrayList<>();
        for (Object attribut : attributs) {
            list.add(attribut);
        }
        return list;
    }

}
```

Classe FabriqueCondition

```
package fabriques;
```

```java
import biens.ABien;
import encheres.interfaces.ICondition;
import encheres.interfaces.fabriques.IFabriqueCondition;

import java.util.ArrayList;
import java.util.List;
import java.util.Locale;

/**
* Modelise une fabrique de plusieurs conditions.
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public class FabriqueCondition implements IFabriqueCondition {

    @Override
    public List<ICondition> fabriqueCondition(String type, ICondition...
attributs) {

        List<ICondition> conditions = getAttributs(attributs);

        type.toLowerCase(Locale.ROOT);

        switch (type) {

            case "habitation":
            case "vehicule" : {
                ABien.setConditions(conditions);
                return conditions;
            }
            default:
                throw new IllegalArgumentException("Type n'est pas defini");

        }
    }

    /**
     * Permet de recuperer les attributs donc les conditions
     * @param attributs les attributs correspondant aux conditions
     * @return la liste de conditions passe en parametre
     */
    private List<ICondition> getAttributs(ICondition... attributs) {
        List<ICondition> list = new ArrayList<>();
        for (ICondition attribut : attributs) {
            list.add(attribut);
        }
        return list;
    }

}
```

Classe FabriqueFraisGestion

```java
package fabriques;

import encheres.interfaces.IFraisGestion;
import encheres.interfaces.fabriques.IFabriqueFraisGestion;
import fraisGestion.FraisGestion10et5;
import fraisGestion.FraisGestion500et1000;

import java.util.Locale;

/**
* Modelise une fabrique d'une regle de frais de gestion.
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public class FabriqueFraisGestion implements IFabriqueFraisGestion {


    @Override
    public IFraisGestion fabriqueFraisGestion(String type) {

        type.toLowerCase(Locale.ROOT);

        switch (type) {

            case "fraisgestion10et5": {
                return new FraisGestion10et5();
            }
            case "fraisgestion500et1000" : {
                return new FraisGestion500et1000();
            }
            default:
                throw new IllegalArgumentException("Type n'est pas defini");

        }

    }


}
```

Classe FabriqueUtilisateur

```java
package fabriques;

import encheres.interfaces.IUtilisateur;
import encheres.interfaces.fabriques.IFabriqueUtilisateur;
import utilisateurs.Client;
import utilisateurs.Employe;
import utilisateurs.Responsable;
```

```java
/**
* Modelise une fabrique d'un utilisateur.
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public class FabriqueUtilisateur implements IFabriqueUtilisateur {

    @Override
    public IUtilisateur fabriqueUtilisateur(String type, String nomUtilisateur) {

        if (nomUtilisateur == null)
            throw new IllegalArgumentException("Le nom utilisateur n'est pas
defini");

        switch (type) {

            case "client": {
                return new Client(nomUtilisateur);
            }
            case "employe" : {
                return new Employe(nomUtilisateur, new FabriqueBien());
            }
            case "responsable" : {
                return new Responsable(nomUtilisateur);
            }
            default:
                throw new IllegalArgumentException("Type n'est pas defini");

        }
    }

}
```

Package fraisGestion

Classe FraisGestion10et5

```java
package fraisGestion;

import encheres.interfaces.IBien;
import encheres.interfaces.IFraisGestion;

import java.util.Calendar;

/**
* Modelise une regle de frais de gestion concrete.
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
```

```java
public class FraisGestion10et5 implements IFraisGestion {

    @Override
    public double fraisGestionActuel(IBien bien) {
        // Si le bien est vendu
        if (bien.getDateF().before(Calendar.getInstance().getTime()) &&
!bien.getSurencheresEnregistrees().isEmpty())
            return bien.getMontant()*0.10;
        else
            return bien.getMontant()*0.05;
    }

}
```

Package utilisateurs

Classe AUtilisateur

```java
package utilisateurs;

import appli.BiensDeApplication;
import encheres.EncherirNotPossibleException;
import encheres.interfaces.IBien;
import encheres.interfaces.IUtilisateur;

import java.util.HashMap;
import java.util.List;

/**
* Modelise une utilisateur en general.
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public abstract class AUtilisateur implements IUtilisateur {

    /**
     * Represente le nom d'utilisateur
     */
    private String nomUtilisateur;

    /**
     * Represente les biens surencheris par l'utilisateur
     */
    private HashMap<IBien, Double> biensSurencheris;

    /**
```

```
    * Represente la liste des biens acquis par cette utilisateur
    */
   private HashMap<IBien, Double> biensAchetes;

   /**
    * Constructeur d'un utilisateur à partir d'un nom d'utilisateur
    * @param nomUtilisateur le nom d'utilisateur de l'utilisateur
    */
   public AUtilisateur(String nomUtilisateur) {
       this.nomUtilisateur = nomUtilisateur;
       this.biensSurencheris = new HashMap<>();
       this.biensAchetes = new HashMap<>();
   }

   // Getters

   @Override
   public String getNomUtilisateur() {
       return nomUtilisateur;
   }

   @Override
   public HashMap<IBien, Double> getBiensSurencheris() {
       return biensSurencheris;
   }

   @Override
   public HashMap<IBien, Double> getBiensAchetes() {
       return biensAchetes;
   }


   // Methods

   @Override
   public List<IBien> consulterBiens() {
       return BiensDeApplication.getBiens();
   }

   @Override
   public void surencherir(IBien bien, double montant) throws
EncherirNotPossibleException {

       bien.encherir(montant);
       biensSurencheris.put(bien, montant);
       bien.setSurencheresEnregistrees(this, montant);

   }

}
```

Classe Client

```java
package utilisateurs;

import encheres.interfaces.IBien;

/**
* Modelise un client.
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public class Client extends AUtilisateur {

    /**
     * Constructeur d'un client à partir d'un nom de client
     * @param nomUtilisateur le nom du client
     */
    public Client(String nomUtilisateur) {
        super(nomUtilisateur);
    }

    @Override
    public IBien inscrireBien(String type, Object... attributs) {
        throw new RuntimeException("Le client ne peut pas inscrire de bien.");
    }

    @Override
    public double consulterFraisGestion(IBien bien) {
        throw new RuntimeException("Le client ne peut pas consulter les frais de
gestion.");
    }

}
```

Classe Employe

```java
package utilisateurs;

import appli.BiensDeApplication;
import encheres.interfaces.IBien;
import encheres.interfaces.fabriques.IFabriqueBien;

/**
* Modelise un employe.
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public class Employe extends AUtilisateur {

    /**
     * Represente une fabrique de bien car l'employe a la possibilite de mettre un
```

```
bien dans l'app
     */
    private IFabriqueBien fabrique;

    /**
     * Constructeur d'un employe à partir d'un nom d'employe et d'une fabrique de
bien
     * @param nomUtilisateur le nom de l'employe
     * @param fabrique la fabrique correspondant a la fabrique de bien
     */
    public Employe(String nomUtilisateur, IFabriqueBien fabrique) {
        super(nomUtilisateur);
        this.fabrique = fabrique;
    }

    @Override
    public IBien inscrireBien(String type, Object... attributs) {
        IBien bienCreer = fabrique.fabriqueBien(type, attributs);

        BiensDeApplication.ajouterBien(bienCreer);

        return bienCreer;
    }

    @Override
    public double consulterFraisGestion(IBien bien) {
        throw new RuntimeException("L'employe ne peut pas consulter les frais de
gestion.");
    }

}
```

Classe Responsable

```
package utilisateurs;

import encheres.interfaces.IBien;

/**
* Modelise un responsable.
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public class Responsable extends AUtilisateur {

    /**
     * Constructeur d'un responsable à partir d'un nom de responsable
     * @param nomUtilisateur le nom du responsable
     */
    public Responsable(String nomUtilisateur) {
```

```
        super(nomUtilisateur);
    }

    @Override
    public double consulterFraisGestion(IBien bien) {
        return bien.consulterFraisGestion();
    }

    @Override
    public IBien inscrireBien(String type, Object... attributs) {
        throw new RuntimeException("Le lardon ne peut pas inscrire de bien.");
    }

}
```

Package interfaceGraphiqueEnchère

Classe BienController

```
package interface_enchere;

import encheres.interfaces.IBien;

import java.util.Date;

public class BienController {
    private String description;
    private double montantD;
    private double montant;
    private Date dateD;
    private Date dateF;
    private IBien bien;

    public BienController(IBien bien) {
        this.description = bien.getDescription();
        this.montantD = bien.getMontantD();
        this.montant = bien.getMontant();
        this.dateD = bien.getDateD();
        this.dateF = bien.getDateF();
        this.bien = bien;
    }

    public String getDescription() {
        return description;
    }

    public double getMontantD() {
```

```
        return montantD;
    }

    public double getMontant() {
        return montant;
    }

    public Date getDateD() {
        return dateD;
    }

    public Date getDateF() {
        return dateF;
    }

    public IBien getBien() {
        return bien;
    }
}
```

Classe Controller

```
package interface_enchere;

import appli.Application;
import appli.BiensDeApplication;
import encheres.EncherirNotPossibleException;
import encheres.interfaces.IBien;
import encheres.interfaces.fabriques.IFabriqueBien;
import fabriques.FabriqueBien;
import fraisGestion.FraisGestion10et5;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.DatePicker;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;
import javafx.scene.control.cell.PropertyValueFactory;

import java.net.URL;
import java.time.ZoneId;
import java.util.Calendar;
import java.util.Date;
import java.util.ResourceBundle;

/**
```

```java
* Le controller de notre interface graphique d'enchere.
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public class Controller {
    @FXML
    private ResourceBundle resources;

    @FXML
    private URL location;

    @FXML
    private TextField montantEnchere;

    @FXML
    private TextField nomEncherisseur;

    @FXML
    private Button surencherir;

    @FXML
    private Button ajouterBien;

    @FXML
    private DatePicker ajtDateDebut;

    @FXML
    private DatePicker ajtDateFin;

    @FXML
    private TextField ajtDescription;

    @FXML
    private TextField ajtMontant;

    @FXML
    private TextField ajtTypeBien;

    @FXML
    private TextField ajtStr;

    @FXML
    private TextField ajtInt;

    @FXML
    private TableView<BienController> listeBiens;

    @FXML
    private TableColumn<BienController, String> listeBiensDescription;

    @FXML
    private TableColumn<BienController, Double> listeBiensMontant;

    @FXML
    private void surencherir() throws EncherirNotPossibleException {
        if(listeBiens.getSelectionModel().isEmpty()) {
            System.out.println("Veuillez sélectionner un bien.");
```

```
            return;
        }
        IBien bienSelectionne =
listeBiens.getSelectionModel().selectedItemProperty().get().getBien();

        double montant;
        try {
            montant = Double.parseDouble(montantEnchere.getText());
        } catch (Exception e) {
            System.out.println("Veuillez entrer une valeur.");
            return;
        }

        bienSelectionne.encherir(montant);
        actualiser();
    }

    @FXML
    private void ajouterBien() {
        assert !ajtTypeBien.getText().isEmpty();
        assert !ajtDescription.getText().isEmpty();
        assert !ajtMontant.getText().isEmpty();
        assert ajtDateDebut.getValue() != null;
        assert ajtDateFin.getValue() != null;
        assert !ajtStr.getText().isEmpty();
        assert !ajtInt.getText().isEmpty();

        String typeBien = ajtTypeBien.getText();
        String description = ajtDescription.getText();

        double montant;
        try {
            montant = Double.parseDouble(ajtMontant.getText());
        } catch (Exception e) {
            System.out.println("Veuillez entrer une valeur.");
            return;
        }


        // Get dateD
        Calendar dateD = Calendar.getInstance();
        try {
            Date dateD_date =
Date.from(ajtDateDebut.getValue().atStartOfDay(ZoneId.systemDefault()).toInstant(
));
            dateD.setTime(dateD_date);
        } catch (Exception e) {
            System.out.println("La date de début n'est pas bonne.");
            return;
        }


        // Get dateD
        Calendar dateF = Calendar.getInstance();
        try {
            Date dateF_date =
Date.from(ajtDateFin.getValue().atStartOfDay(ZoneId.systemDefault()).toInstant())
```

```
;
            dateF.setTime(dateF_date);
        } catch (Exception e) {
            System.out.println("La date de fin n'est pas bonne.");
            return;
        }


        String firstSpecialAttribute = ajtStr.getText();

        int secondSpecialAttribute;
        try {
            secondSpecialAttribute = Integer.parseInt(ajtInt.getText());
        } catch (Exception e) {
            System.out.println("Veuillez entrer une valeur.");
            return;
        }

        IFabriqueBien fb = new FabriqueBien();

        try {
            BiensDeApplication.ajouterBien(fb.fabriqueBien(typeBien, description,
montant, dateD, dateF,
                    new FraisGestion10et5(), firstSpecialAttribute,
secondSpecialAttribute));
        } catch (Exception e) {
            System.out.println("Des valeurs sont manquantes ou incorrectes");
            return;
        }


        actualiser();
    }

    @FXML
    void initialize() {
        Application.main(null);

        listeBiensDescription.setCellValueFactory(new
PropertyValueFactory<>("description"));
        listeBiensMontant.setCellValueFactory(new
PropertyValueFactory<>("montant"));

        actualiser();
    }

    public void actualiser() {
        ObservableList liste_person = FXCollections.observableArrayList();
        for( IBien bien : BiensDeApplication.getBiens() ){
            liste_person.add(new BienController(bien));
        }
        listeBiens.setItems(liste_person);
    }
}
```

Classe Main

```
package interface_enchere;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.stage.Stage;

import java.io.File;
import java.net.MalformedURLException;


/**
* La classe main qui lance l'application.
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public class Main extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception{
        Parent root = FXMLLoader.load(getClass().getResource("sample.fxml"));

        primaryStage.getIcons().add(new Image("ventes-encheres-immobilier.jpg"));
        primaryStage.setTitle("Enchère");
        Scene scene = new Scene(root, 600, 600);
        String fontSheet = fileToStylesheetString( new
File("src/interface_enchere/stylesheet.css") );

        scene.getStylesheets().add( fontSheet );

        primaryStage.setScene(scene);
        primaryStage.setResizable(false);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }

    public String fileToStylesheetString ( File stylesheetFile ) {
        try {
            return stylesheetFile.toURI().toURL().toString();
        } catch ( MalformedURLException e ) {
            return null;
        }
    }

}
```

FXML sample.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.DatePicker?>
<?import javafx.scene.control.TableColumn?>
<?import javafx.scene.control.TableView?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.text.Font?>
<?import javafx.scene.text.Text?>

<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
minWidth="-Infinity" prefHeight="600.0" prefWidth="600.0"
stylesheets="@stylesheet.css" xmlns="http://javafx.com/javafx/17"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="interface_enchere.Controller">
  <children>
     <Text fill="#14bee3" layoutX="94.0" layoutY="49.0" strokeType="OUTSIDE"
strokeWidth="0.0" text="BIENVENUE SUR L'APPLICATION &quot;ENCHÈRES&quot;"
underline="true">
        <font>
           <Font name="System Bold Italic" size="19.0" />
        </font>
     </Text>
     <VBox layoutX="14.0" layoutY="94.0" prefHeight="200.0" prefWidth="569.0">
        <children>
           <TableView fx:id="listeBiens" prefHeight="200.0" prefWidth="200.0">
             <columns>
               <TableColumn fx:id="listeBiensDescription"
prefWidth="403.9999694824219" text="Description du bien" />
               <TableColumn fx:id="listeBiensMontant"
prefWidth="164.66668701171875" text="Montant du bien" />
             </columns>
           </TableView>
        </children>
     </VBox>
     <TextField fx:id="ajtTypeBien" layoutX="17.0" layoutY="503.0"
prefHeight="25.0" prefWidth="96.0" promptText="Type de bien" />
     <TextField fx:id="ajtDescription" layoutX="125.0" layoutY="503.0"
prefHeight="25.0" prefWidth="351.0" promptText="Description" />
     <TextField fx:id="ajtMontant" layoutX="485.0" layoutY="503.0"
prefHeight="25.0" prefWidth="100.0" promptText="Montant" />
     <DatePicker fx:id="ajtDateDebut" layoutX="16.0" layoutY="544.0"
prefHeight="25.0" prefWidth="100.0" promptText="Début" />
     <TextField fx:id="ajtStr" layoutX="253.0" layoutY="544.0" prefHeight="25.0"
prefWidth="113.0" promptText="Ville/Marque" />
     <DatePicker fx:id="ajtDateFin" layoutX="125.0" layoutY="544.0"
```

```
prefHeight="25.0" prefWidth="113.0" promptText="Fin" />
      <TextField fx:id="ajtInt" layoutX="385.0" layoutY="544.0" prefHeight="25.0"
prefWidth="113.0" promptText="Nb Pièces/Année" />
      <Button fx:id="ajouterBien" layoutX="509.0" layoutY="544.0"
mnemonicParsing="false" onAction="#ajouterBien" prefHeight="25.0"
prefWidth="75.0" text="Ajouter" />
      <Text fill="#14bee3" layoutX="243.0" layoutY="480.0" strokeType="OUTSIDE"
strokeWidth="0.0" text="Ajouter un bien" underline="true">
         <font>
            <Font name="System Bold Italic" size="16.0" />
         </font>
      </Text>
      <Text fill="#14bee3" layoutX="258.0" layoutY="331.0" strokeType="OUTSIDE"
strokeWidth="0.0" text="Surenchérir" underline="true">
         <font>
            <Font name="System Bold Italic" size="16.0" />
         </font>
      </Text>
      <TextField fx:id="nomEncherisseur" layoutX="51.0" layoutY="359.0"
promptText="Nom de l'enchérisseur" />
      <TextField fx:id="montantEnchere" layoutX="253.0" layoutY="359.0"
promptText="Montant de l'enchère" />
      <Button fx:id="surencherir" layoutX="459.0" layoutY="359.0"
mnemonicParsing="false" onAction="#surencherir" prefHeight="25.0"
prefWidth="96.0" text="Surenchérir" />
   </children>
</AnchorPane>
```

CSS Stylesheet.css

```
.root {
    -fx-background-color: #ffc2de;
}
```

Package tests

Classe BienTest

```java
import appli.BiensDeApplication;
import conditions.IsMontantMaxValidVehicule;
import conditions.IsMontantMinValidHabitation;
import conditions.IsMontantMinValidVehicule;
import conditions.IsPeriodeValid;
import encheres.EncherirNotPossibleException;
import encheres.interfaces.IBien;
import encheres.interfaces.IFraisGestion;
import encheres.interfaces.IUtilisateur;
import encheres.interfaces.fabriques.IFabriqueBien;
import encheres.interfaces.fabriques.IFabriqueCondition;
import encheres.interfaces.fabriques.IFabriqueFraisGestion;
import encheres.interfaces.fabriques.IFabriqueUtilisateur;
import fabriques.FabriqueBien;
import fabriques.FabriqueCondition;
import fabriques.FabriqueFraisGestion;
import fabriques.FabriqueUtilisateur;
import fraisGestion.FraisGestion10et5;
import fraisGestion.FraisGestion500et1000;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import java.util.*;

import static org.junit.jupiter.api.Assertions.*;

/**
* Les tests sur notre application d'enchere.
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public class BienTest {

    private IFabriqueBien fabriqueBien;
    private IFabriqueFraisGestion fabriqueFraisGestion;
    private IFabriqueUtilisateur fabriqueUtilisateur;
    private IFabriqueCondition fabriqueCondition;
    private Calendar dateD;
    private Calendar dateF;
    private IFraisGestion fraisGestion;
    private IUtilisateur c;
    private IUtilisateur r;
    private IUtilisateur e;
    private IBien habitation;
    private IBien vehicule;

    @BeforeEach
    void initialisationDonnee() {

        // Differentes Fabriques
        fabriqueBien = new FabriqueBien();
        fabriqueFraisGestion = new FabriqueFraisGestion();
        fabriqueUtilisateur = new FabriqueUtilisateur();
        fabriqueCondition = new FabriqueCondition();

        // Dates
        dateD = new GregorianCalendar(2021, Calendar.OCTOBER, 1);
```

```
        dateF = new GregorianCalendar(2021, Calendar.DECEMBER, 1);

        // Frais Gestion
        fraisGestion =
fabriqueFraisGestion.fabriqueFraisGestion("fraisgestion10et5");

        // Les Utilisateurs
        c = fabriqueUtilisateur.fabriqueUtilisateur("client", "Jeff Bezos");
        r = fabriqueUtilisateur.fabriqueUtilisateur("responsable", "Coralie");
        e = fabriqueUtilisateur.fabriqueUtilisateur("employe", "Hervé");

        // Les Biens
        habitation = fabriqueBien.fabriqueBien(
                "habitation",
                "Un appartement dans le 16ème Arrondissement", 250000.0, dateD,
dateF, fraisGestion,
                "Paris", 5);
        vehicule = fabriqueBien.fabriqueBien(
                "vehicule",
                "Une Tesla Model X", 55000.0, dateD, dateF, fraisGestion, "Tesla",
2021);
    }

    @Test
    void clientConsulterFraisGestionImpossible() {
        // When
        assertThrows(RuntimeException.class, () ->
c.consulterFraisGestion(habitation));
    }

    @Test
    void employeConsulterFraisGestionImpossible() {
        // When
        assertThrows(RuntimeException.class, () ->
e.consulterFraisGestion(habitation));
    }

    @Test
    void responsableConsulterFraisGestionImpossible() {
        // When
        assertThrows(RuntimeException.class, () -> r.inscrireBien("habitation",
"habitation",
                "Un appartement dans le 16ème Arrondissement", 250000.0, dateD,
dateF, fraisGestion, "Paris", 5));
    }

    @Test
    void surrencherirHabitationMontantEtPeriodeConforme() {
        // Given
        fabriqueCondition.fabriqueCondition("habitation", new
IsMontantMinValidHabitation(), new IsPeriodeValid());

        // When
        assertDoesNotThrow(() -> e.surencherir(habitation, 300000.0));
    }

    @Test
```

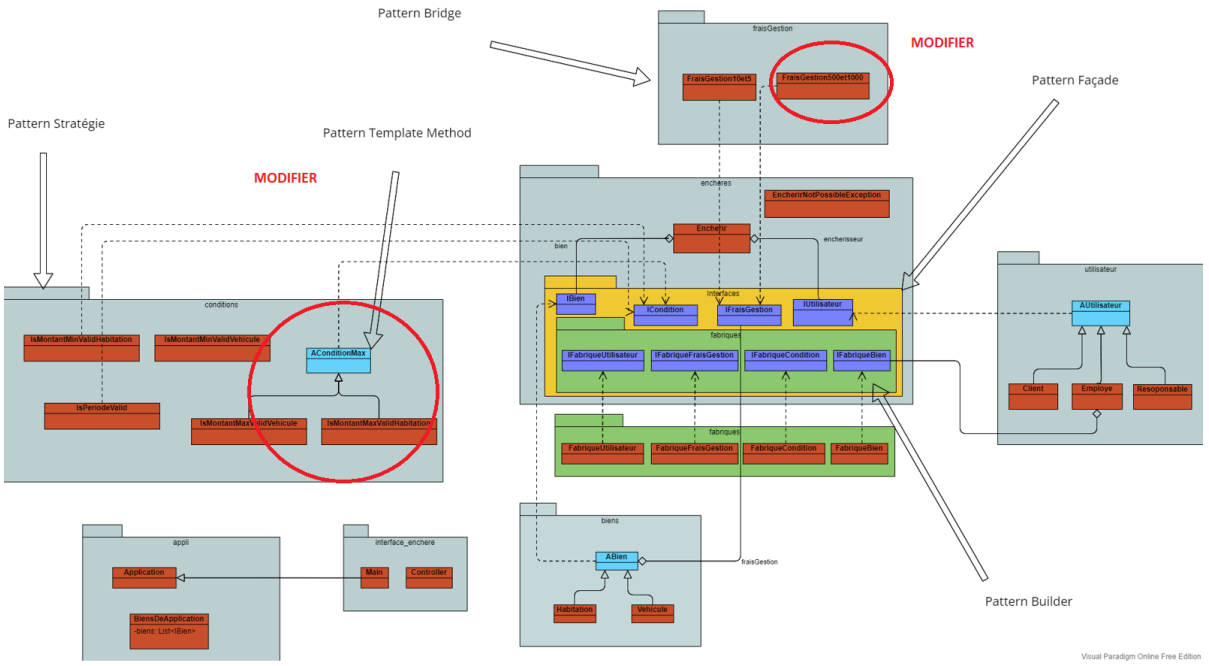
```
    void surrencherirHabitationPeriodePasConforme() {
        // Given
        dateF = new GregorianCalendar(2021, Calendar.NOVEMBER, 1);
        IBien habitation2;
        fabriqueCondition.fabriqueCondition("habitation", new
IsMontantMinValidHabitation(), new IsPeriodeValid());

        // When : changing the end date
        habitation2 = e.inscrireBien("habitation",
                "Un appartement dans le 16ème Arrondissement", 250000.0, dateD,
                dateF,
                new FraisGestion10et5(), "Paris", 5);

        assertThrows(EncherirNotPossibleException.class, () ->
e.surencherir(habitation2, 300000.0));

        // Then : Rien car pas de resultat

    }

    @Test
    void surrencherirHabitationMontantPasConforme() {
        // Given
        fabriqueCondition.fabriqueCondition("habitation", new
IsMontantMinValidHabitation(), new IsPeriodeValid());

        // When
        assertThrows(EncherirNotPossibleException.class, () ->
e.surencherir(habitation, 200000.0));
    }

    @Test
    void surrencherirVehiculeMontantEtPeriodeConforme() {
        // Given
        fabriqueCondition.fabriqueCondition("vehicule", new
IsMontantMinValidVehicule(), new IsPeriodeValid());

        // When
        assertDoesNotThrow(()->e.surencherir(vehicule, 60000.0));
    }

    @Test
    void surrencherirVehiculePeriodePasConforme() {
        // Given
        dateF = new GregorianCalendar(2021, Calendar.NOVEMBER, 1);
        IBien vehicule2;
        fabriqueCondition.fabriqueCondition("vehicule", new
IsMontantMinValidVehicule(), new IsPeriodeValid());

        // When : changing the end date
        vehicule2 = fabriqueBien.fabriqueBien("vehicule",
                "Une Tesla Model X", 55000.0, dateD,
                dateF,
                new FraisGestion10et5(), "Tesla", 2021);

        assertThrows(EncherirNotPossibleException.class, () ->
e.surencherir(vehicule2, 60000.0));
```

```
        // Then : Rien car pas de resultat

    }

    @Test
    void surrencherirVehiculeMontantPasConforme() {
        // Given
        fabriqueCondition.fabriqueCondition("vehicule", new
IsMontantMinValidVehicule(), new IsPeriodeValid());

        // Then
        assertThrows(EncherirNotPossibleException.class, ()->
e.surencherir(vehicule, 45000.0));
    }

    @Test
    void consulterFraisDeGestionDunBienNonVendu() {
        // Given
        double expectedFraisGestion = 12500.0; // 5 % pour un bien non vendu

        // When
        double actualFraisGestion = r.consulterFraisGestion(habitation);

        // Then
        assertEquals(expectedFraisGestion, actualFraisGestion);
    }

    @Test
    void consulterBiensMisAuEnchereCorrect() {
        // Given
        BiensDeApplication.clearListBiens(); // reset la liste des biens à vide
        List<IBien> expectedBiens = new ArrayList<>();
        expectedBiens.add(vehicule); expectedBiens.add(habitation);

        // When
        BiensDeApplication.ajouterBien(vehicule);
        BiensDeApplication.ajouterBien(habitation);
        List<IBien> actualBiens = c.consulterBiens();

        // Then
        assertEquals(expectedBiens, actualBiens);
    }

    @Test
    void consulterBiensMisAuEnchereIncorrect() {
        // Given
        BiensDeApplication.clearListBiens(); // reset la liste des biens à vide
        List<IBien> expectedBiens = new ArrayList<>();
        expectedBiens.add(vehicule); expectedBiens.add(habitation);

        // When
        BiensDeApplication.ajouterBien(vehicule);
        List<IBien> actualBiens = c.consulterBiens();

        // Then
        assertNotEquals(expectedBiens, actualBiens);
```

```
    }

    @Test
    void consulterSurencheresEnregistrerHabitation() throws
EncherirNotPossibleException {
        // Given
        HashMap<IUtilisateur, Double> expectedSEnregistrer = new HashMap<>();
        expectedSEnregistrer.put(c, 300000.0); expectedSEnregistrer.put(c,
350000.0);

        // When
        c.surencherir(habitation, 300000.0);
        c.surencherir(habitation, 350000.0);
        HashMap<IUtilisateur, Double> actualSEnregistrer =
habitation.getSurencheresEnregistrees();

        // Then
        assertEquals(expectedSEnregistrer, actualSEnregistrer);
    }

    @Test
    void consulterSurencheresEnregistrerVehicule() throws
EncherirNotPossibleException {
        // Given
        IUtilisateur c1 = fabriqueUtilisateur.fabriqueUtilisateur("client", "Bill
Gates");
        HashMap<IUtilisateur, Double> expectedSEnregistrer = new HashMap<>();
        expectedSEnregistrer.put(c1, 300000.0); expectedSEnregistrer.put(c1,
300001.0);
        fabriqueCondition.fabriqueCondition("vehicule", new
IsMontantMinValidVehicule(), new IsPeriodeValid());

        // When
        c1.surencherir(vehicule, 300000.0);
        c1.surencherir(vehicule, 300001.0);
        HashMap<IUtilisateur, Double> actualSEnregistrer =
vehicule.getSurencheresEnregistrees();

        // Then
        assertEquals(expectedSEnregistrer, actualSEnregistrer);
    }

}
```

Seconde Itération :

Diagramme de classe:



Code Java :

Classe AConditionMax

```java
package conditions;

import encheres.interfaces.IBien;
import encheres.interfaces.ICondition;

public abstract class AConditionMax implements ICondition {
    @Override
    public boolean isConditionRespected(IBien bien, double montant) {
        return montant <= bien.getMontant() * getPourcentageMax();
    }

    /**
     * Permet de changer le pourcentage maximal.
     * @param pourcentageMax le nouveau pourcentage maximal.
     */
    protected abstract void setPourcentageMax(double pourcentageMax);

    /**
     * Permet de retourner le pourcentage maximal.
     * @return le pourcentage maximal.
     */
    protected abstract double getPourcentageMax();
}
```

Classe IsMontantMaxValidHabitation

```java
package conditions;

import encheres.interfaces.IBien;
import encheres.interfaces.ICondition;

/**
* Modélise une condition de l'enchere d'une habitation
* permettant la verification du mantant a ne pas exceder
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public class IsMontantMaxValidHabitation extends AConditionMax {
    /**
     * Correspond au pourcentage maximal autorisé pour une enchère.
     */
    private double pourcentageMax = 1.25;

    @Override
    public double getPourcentageMax() {
        return pourcentageMax;
    }

    @Override
    public void setPourcentageMax(double pourcentageMax) {
        this.pourcentageMax = pourcentageMax;
    }
}
```

Classe IsMontantMaxValidVehicule

```java
package conditions;

import encheres.interfaces.IBien;
import encheres.interfaces.ICondition;

/**
* Modélise une condition de l'enchere d'un vehicule
```

```
* permettant la verification du mantant a ne pas exceder
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public class IsMontantMaxValidVehicule extends AConditionMax {
    /**
     * Correspond au pourcentage maximal autorisé pour une enchère.
     */
    private double pourcentageMax = 1.2;

    @Override
    public double getPourcentageMax() {
        return pourcentageMax;
    }

    @Override
    public void setPourcentageMax(double pourcentageMax) {
        this.pourcentageMax = pourcentageMax;
    }
}
```

Classe FraisGestion500et1000

```
package fraisGestion;

import biens.Habitation;
import biens.Vehicule;
import encheres.interfaces.IBien;
import encheres.interfaces.IFraisGestion;

/**
* Modelise une regle de frais de gestion concrete.
* @author  Martin-Dipp Daryl, Maxime Wang
* @version 1.0
*/
public class FraisGestion500et1000 implements IFraisGestion {

    @Override
    public double fraisGestionActuel(IBien bien) {
        if (bien instanceof Habitation)
            return 1000;
        else if (bien instanceof Vehicule)
            return 500;
        else
            throw new IllegalArgumentException("Le bien n'a pas de frais de
gestion.");
    }
```

```
}
```

Classe BienTest

```
    @Test
    void consulterFraisDeGestionDuneHabitation() {
        // Given
        double expectedFraisGestion = 1000.0;
        habitation.setFraisGestion(new FraisGestion500et1000());

        // When
        double actualFraisGestion = r.consulterFraisGestion(habitation);

        // Then
        assertEquals(expectedFraisGestion, actualFraisGestion);
    }

    @Test
    void consulterMauvaisFraisDeGestionDuneHabitation() {
        // Given
        double expectedFraisGestion = 100.0;
        habitation.setFraisGestion(new FraisGestion500et1000());

        // When
        double actualFraisGestion = r.consulterFraisGestion(habitation);

        // Then
        assertNotEquals(expectedFraisGestion, actualFraisGestion);
    }

    @Test
    void consulterFraisDeGestionDunVehicule() {
        // Given
        double expectedFraisGestion = 500.0;
        vehicule.setFraisGestion(new FraisGestion500et1000());

        // When
        double actualFraisGestion = r.consulterFraisGestion(vehicule);

        // Then
        assertEquals(expectedFraisGestion, actualFraisGestion);
    }

    @Test
    void consulterMauvaisFraisDeGestionDunVehicule() {
        // Given
        double expectedFraisGestion = 50.0;
        vehicule.setFraisGestion(new FraisGestion500et1000());

        // When
```

```
        double actualFraisGestion = r.consulterFraisGestion(vehicule);

        // Then
        assertNotEquals(expectedFraisGestion, actualFraisGestion);
    }

    @Test
    void surrencherirHabitationMontantMaxConforme() {
        // Given
        fabriqueCondition.fabriqueCondition("habitation", new
IsMontantMinValidVehicule(), new IsPeriodeValid(),
                new IsMontantMaxValidVehicule());

        // Then
        assertDoesNotThrow(() -> e.surencherir(habitation, 300000.0));
    }

    @Test
    void surrencherirHabitationMontantMaxPasConforme() {
        // Given
        fabriqueCondition.fabriqueCondition("habitation", new
IsMontantMinValidVehicule(), new IsPeriodeValid(),
                new IsMontantMaxValidVehicule());

        // Then
        assertThrows(EncherirNotPossibleException.class, ()->
e.surencherir(habitation, 512545.0));
    }

    @Test
    void surrencherirVehiculeMontantMaxConforme() {
        // Given
        fabriqueCondition.fabriqueCondition("vehicule", new
IsMontantMinValidVehicule(), new IsPeriodeValid(),
                new IsMontantMaxValidVehicule());

        // Then
        assertDoesNotThrow(() -> e.surencherir(vehicule, 55000.0));
    }

    @Test
    void surrencherirVehiculeMontantMaxPasConforme() {
        // Given
        fabriqueCondition.fabriqueCondition("vehicule", new
IsMontantMinValidVehicule(), new IsPeriodeValid(),
                new IsMontantMaxValidVehicule());

        // Then
        assertThrows(EncherirNotPossibleException.class, ()->
e.surencherir(vehicule, 512545.0));
    }

@Test
void changerMontantMax() {
    // Given
    IsMontantMaxValidVehicule isMontantMax = new IsMontantMaxValidVehicule();
```

```
    double pourcentageExpected = isMontantMax.getPourcentageMax();

    fabriqueCondition.fabriqueCondition("vehicule", new
IsMontantMinValidVehicule(), new IsPeriodeValid(),
            isMontantMax);

    // When
    isMontantMax.setPourcentageMax(1.4);
    double pourcentageActual = isMontantMax.getPourcentageMax();

    // Then
    assertNotEquals(pourcentageExpected, pourcentageActual);
}
```