

Fiche d'exercices - Corrections

Questions de cours :

Introduction

1) Citez deux raisons motivant la modélisation d'un système complexe. Expliquez-les.

Choix multiple, dont :

- *Afin de mieux comprendre un système complexe / une partie de la réalité*
- *Afin de représenter de manière simplifier le monde / un système*
- *Afin de communiquer sur une vision / compréhension du monde / d'un système*
- *Afin d'assurer la compréhension d'un système complexe par une représentation intelligible*

2) Définissez la notion de modèle. Quel intérêt avons-nous en construire un modèle ?

- *Un modèle est une représentation simplifiée de la réalité*
- *Représentation abstraite et simplifiée d'une entité*

3) « UML est la méthode de conception de logiciels le plus répandu à nous jours ». Etes-vous d'accord ? Argumenter.

Faux. UML est un langage, pas une méthode de conception.

4) Citez 3 raisons qui motivent l'usage d'UML pour le développement d'un système d'information.

Choix multiple, dont :

- *Structurer les connaissances sur un projet*
- *Aider à documenter un projet*
- *Communiquer autour d'un schéma / d'un modèle*
- *Capitaliser la connaissance autour d'un projet*
- *Générer du code correspondant aux modèles*

Diagramme de classes :

5) Quel intérêt avons-nous d'utiliser les diagrammes de classe ? A quoi servent-ils ?

Les diagrammes de classe servent à identifier les concepts propres au domaine, à montrer la structure interne du système, à avoir une vision statique du système.

6) Quels éléments pouvons-nous représenter à l'intérieur d'un diagramme de classe ? Donnez des exemples.

Éléments possibles dans un diagramme de classe sont les classes (avec leurs attributs et leurs opérations), les associations reliant les classes, les rôles et les multiplicités associés aux associations, les indications sur la visibilité des attributs, des opérations et des rôles, et sur la navigabilité des associations.

- 7) « Les attributs dérivés sont des attributs redondants, qu'on ne peut pas représenter dans un diagramme de classe ». Vrai ou faux ? Justifiez votre réponse.

Faux. Les attributs dérivés, même en étant calculés à partir d'autres attributs, peuvent être représentés dans un diagramme de classe s'ils s'avèrent pertinents pour la compréhension du modèle.

- 8) Quelle est l'importance des associations au sein d'un diagramme de classe ? Donnez deux exemples.

Les associations représentent les relations sémantiques entre les classes. Exemples : Un étudiant est inscrit à une université ; un à plusieurs enseignants enseignent une matière...

- 9) Distinguez une association binaire d'une association n-aire. Donnez un exemple de chaque.

Une association binaire implique deux classes, tandis qu'une association n-aire implique au moins trois classes. Une association binaire serait un étudiant qui est inscrit à une université. Un exemple de n-aire serait un étudiant qui est inscrit à une matière avec un enseignant.

- 10) Que représente la multiplicité au sein d'une association ? Et la multiplicité des attributs dans une classe ? Donnez un exemple de chaque.

La multiplicité associée à une extrémité donnée d'une association indique le nombre d'objets de cette extrémité qui peuvent être liés à un seul objet de l'autre classe. La multiplicité d'un attribut indique les quantités ou le caractère optionnel d'un attribut au sein d'une classe. Exemples :

Etudiant [] ----- inscrit à ----- [1] Université*

Dans la classe étudiant, on peut considérer qu'un étudiant a plusieurs emails : emails : String [0..]*

Diagramme de séquence :

- 11) Quel intérêt avons-nous d'utiliser les diagrammes de séquence ? A quoi servent-ils ?

Les diagrammes de séquence permettent la modélisation des interactions entre les instances d'un modèle dans un angle temporel. Ils représentent l'échange des messages entre les instances dans le temps. Diagrammes de séquence établissent le pont entre les diagrammes de cas d'utilisation et de classe. Ils sont utilisés pour détailler les interactions prévues dans les UC, pour affiner les diagrammes de classe, en précisant les responsabilités, surtout les opérations, aux classes.

- 12) Quel rapport pouvons-nous établir entre les diagrammes de séquence et les diagrammes de classes ? Expliquez.

Les diagrammes de séquence sont utilisés pour détailler les interactions entre les éléments représentés dans les diagrammes de classes. Ils sont utilisés pour affiner les diagrammes de classes, en précisant les responsabilités de chaque élément, et surtout les opérations.

- 13) « Les diagrammes de séquence représentent l'ordre des échanges de messages entre les classes ». Expliquez cette affirmation. Etes-vous d'accord ?

Les diagrammes de séquence représentent surtout l'échange des messages dans le temps, ce qui va impliquer un ordre temporel précis. Par ailleurs, cet échange se déroule surtout entre instances, mais pas uniquement. Par exemple, des acteurs aussi être impliqués dans un diagramme de séquence.

- 14) Que représente-t-il une ligne de vie dans un diagramme de séquence ?

Comme son nom l'indique, une ligne de vie dans un SD représente l'existence d'un participant (normalement, une instance) dans le temps.

- 15) Un objet peut-il envoyer un message à lui-même ?

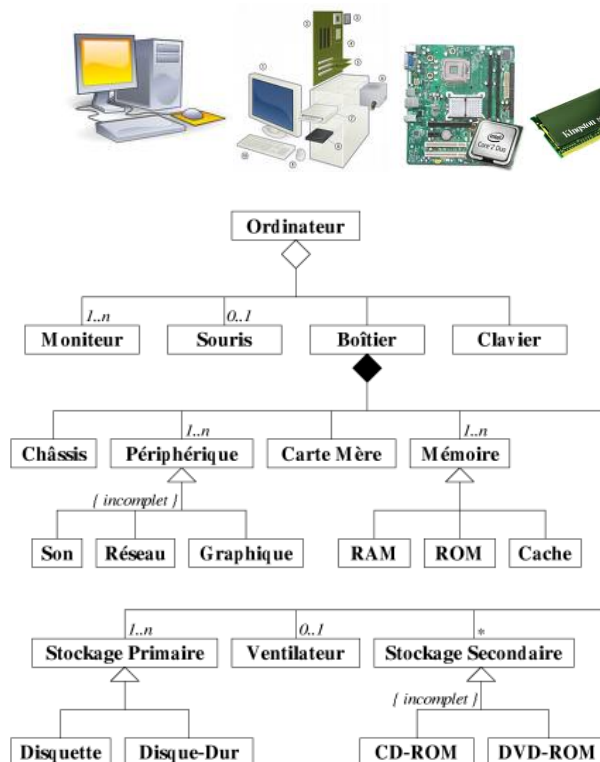
Oui. On peut considérer dans ce cas qu'il s'agit d'un traitement interne.

Exercices de modélisation :

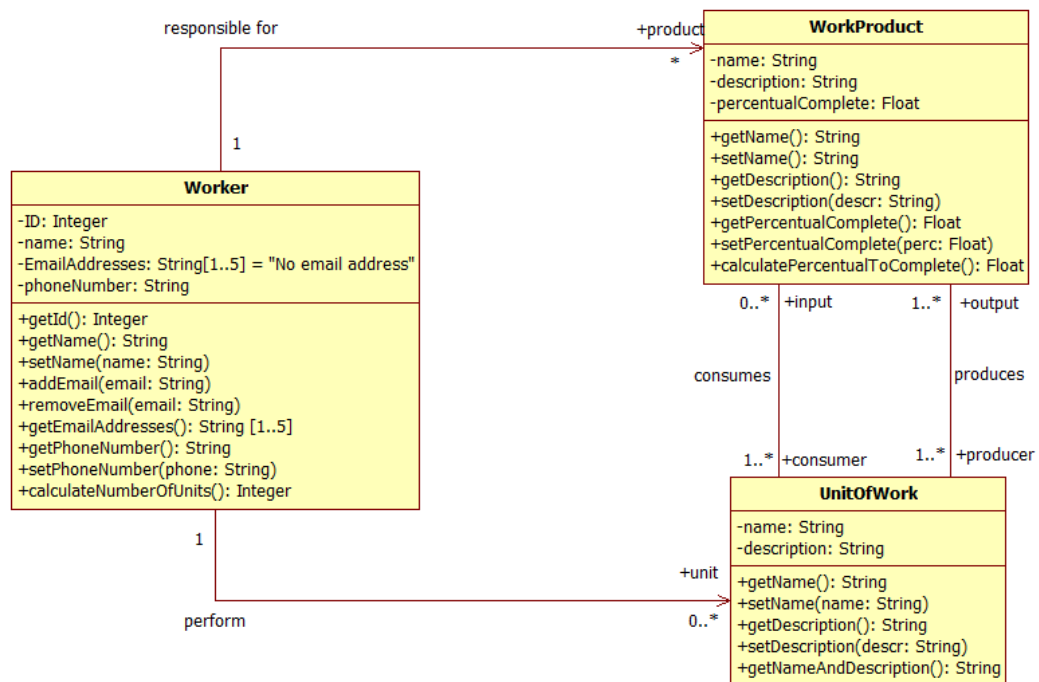
Diagramme de classes :

- 1) Identifiez et modélisez les classes et les associations dans le scénario suivant.

Un ordinateur est composé d'un ou plusieurs moniteurs, d'un boîtier, d'une souris et d'un clavier. Un boîtier a un châssis métallique, une carte mère, plusieurs barrettes de mémoire (RAM, ROM et cache), un ventilateur optionnel, des supports de stockage (disquette, disque-dur, CD-ROM, DVD-ROM...), et des cartes périphériques (son, réseau, graphique...). Un ordinateur possède toujours au moins un lecteur de disquette ou un disque-dur.



2) Répondez aux questions ci-dessous concernant le diagramme de classe ci-après.



a) Quelles sont les classes impliquées ? Quels sont leurs attributs et leurs opérations ?

Trois classes sont impliquées : *Worker*, *WorkProduct* et *UnitOfWork*. La première compte les attributs *ID*, *name*, *emailAddress* et *phoneNumber*, la deuxième les attributs *name*, *description* et *percentualComplete*, et la troisième, les attributs *name* et *description*. Les opérations sont celles pour récupérer et définir les valeurs des attributs, plus l'opération *calculateNumberOfUnits* pour la classe *Worker*, l'opération *calculatePercentualToComplete* pour la classe *WorkProduct*, et l'opération *getNameAndDescription* pour la classe *UnitOfWork*.

b) Quelle est la visibilité des attributs et des opérations définis par ces classes ?

Les attributs sont tous privés, tandis que les opérations sont toutes publiques.

c) Quelles sont les associations représentées dans le diagramme ? Quels rôles sont-ils définis ? Quelle est la multiplicité associée à chaque rôle ? Quelle navigabilité ?

Quatre associations sont définies : « *responsible for* » entre les classes *Worker* et *WorkProduct* ; « *perform* » entre *Worker* et *UnitOfWork*, « *consumes* » et « *produces* » entre les classes *WorkProduct* et *UnitOfWork*. La première compte le rôle « *product* » du côté de la classe *WorkProduct*, avec une multiplicité de 0..*, et une multiplicité de 1 dans l'autre extrémité. La deuxième association compte le rôle « *unit* » du côté de la classe *UnitOfWork*, avec une multiplicité de 0..*, et une multiplicité de 1 dans l'autre extrémité. Dans ces deux associations, la navigabilité va de la classe *Worker* vers les autres extrémités (et non le contraire). L'association « *consumes* » implique les rôles « *input* » dans l'extrémité *WorkProduct*, avec une multiplicité de *, et « *consumer* » dans l'extrémité *UnitOfWork*, avec une multiplicité de 1..*. L'association « *produces* » implique les rôles « *output* » dans l'extrémité

WorkProduit, avec une multiplicité de *, et « producer » dans l'extrémité UnitOfWork, avec une multiplicité de 1..*. Dans ces deux associations, aucune indication de navigabilité n'est donnée.

d) Que pouvons-nous faire pour rendre ce diagramme plus lisible ?

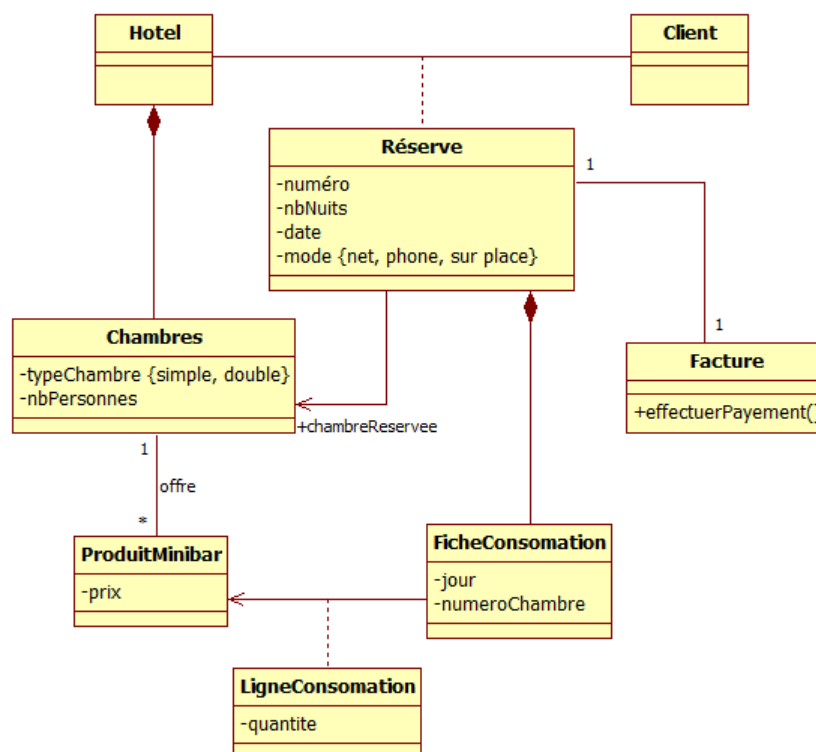
On peut omettre les opérations de manipulation (récupération et définition) des attributs.

3) Proposez un diagramme de classes qui représente le scénario ci-dessous.

L'hôtel 'Bonne Famille' décide de s'informatiser. Il s'agit d'informatiser la gestion des chambres. L'hôtel dispose de 35 chambres : 25 chambres doubles (avec un lit pour deux personnes) et 10 chambres simples (avec un lit pour une personne). Chaque chambre possède un mini-bar avec des boissons (qui sont bien entendu payantes) et des friandises (payantes également).

Chaque matin, la femme de ménage fait le compte des éléments manquants dans le mini-bar et note les informations sur une fiche de consommation. Sur cette fiche se trouve : le n° de la chambre, la date, et, par ligne de consommation : le type de produit consommé et le prix de chaque produit (boisson, friandises, pressing et nettoyage de chaussures).

D'autres modélisations sont également possibles.



4) Le scénario ci-dessous décrit un système d'achat en ligne. Modélisez le système à l'aide d'un diagramme de classes. Séparez les classes, à l'aide des stéréotypes, en trois paquetages types : (i) « Présentation », contenant les classes responsables par les pages Web ; (ii) « Métier », contenant les classes décrivant les données manipulées ; et (ii) « Contrôle », qui fait le lien entre les deux précédents, en gérant le processus de vente en ligne. Représenter cette séparation dans un diagramme de paquetage.

Une entreprise propose un système d'achat en ligne permettant à ses clients l'achat de toute sa gamme de produits. A l'aide de ce système, les clients peuvent gérer leurs achats à travers un panier virtuel. Les clients peuvent comparer les produits qui sont dans leur paniers, ainsi qu'y ajouter ou supprimer des produits. Un panier peut contenir différents produits, ainsi que la date de l'achat. Tous les produits sont identifiés par un nom et un code. Au moment d'ajouter ou de supprimer un produit du panier, le client doit indiquer sa quantité. Trois types de pages Web sont disponibles pour la gestion du panier : une interface Web traditionnelle, où sont utilisées des pages Web aux couleurs paramétrables (l'internaute peut changer les couleurs pendant la visite) ; une page adaptée pour l'accès iPhone non-paramétrable ; et une page conçue spécialement pour une borne interactive située dans l'entrée de l'usine, laquelle permet aussi aux visiteurs de visualiser des vidéos sur l'usine disponible sur YouTube.

D'autres modélisations sont également possible.

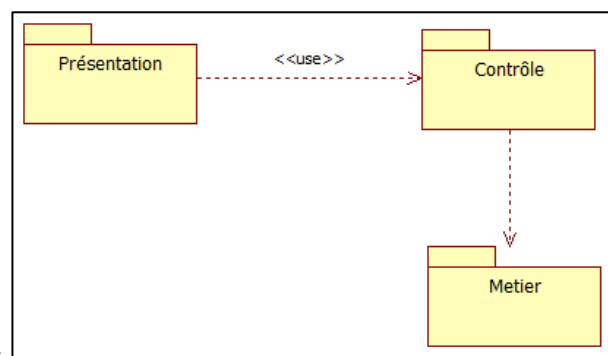
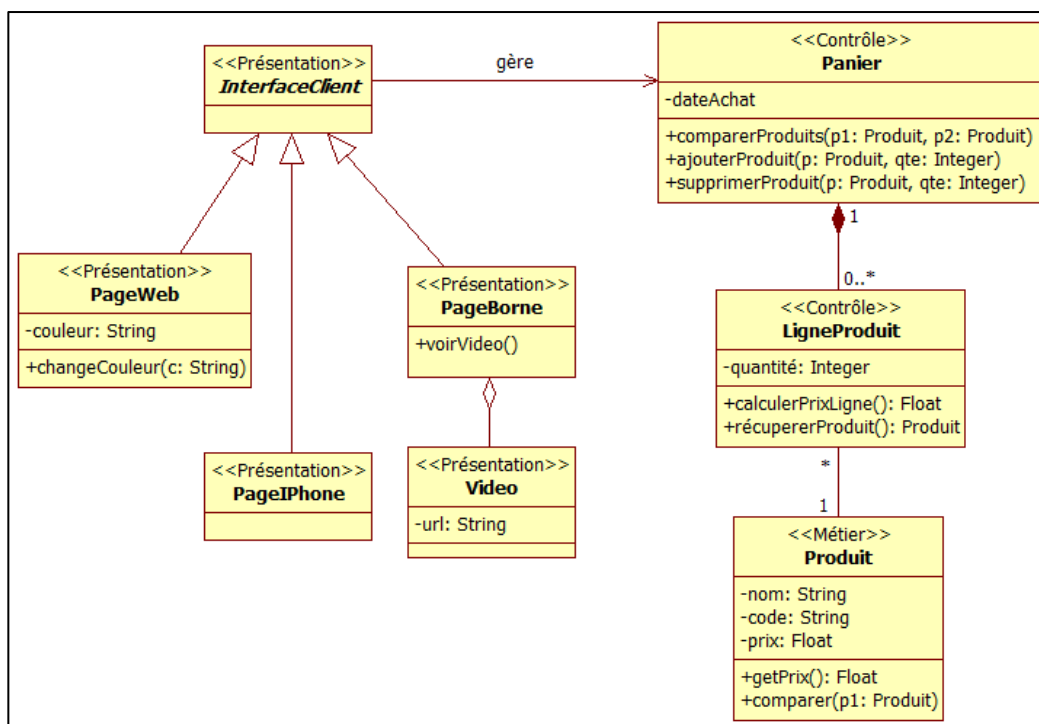
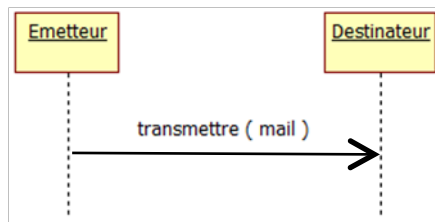


Diagramme de paquetage :

Diagramme de séquence :

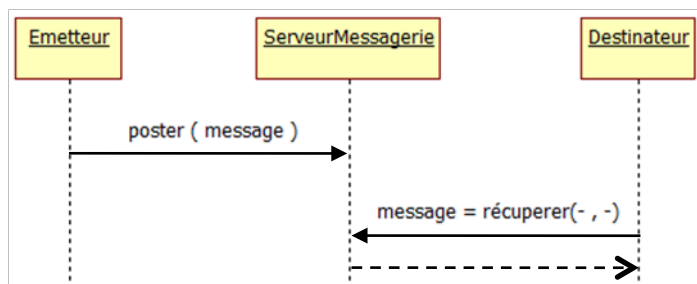
5) Remplissez les diagrammes de séquence ci-dessous.

a) Transmission d'un courrier électronique



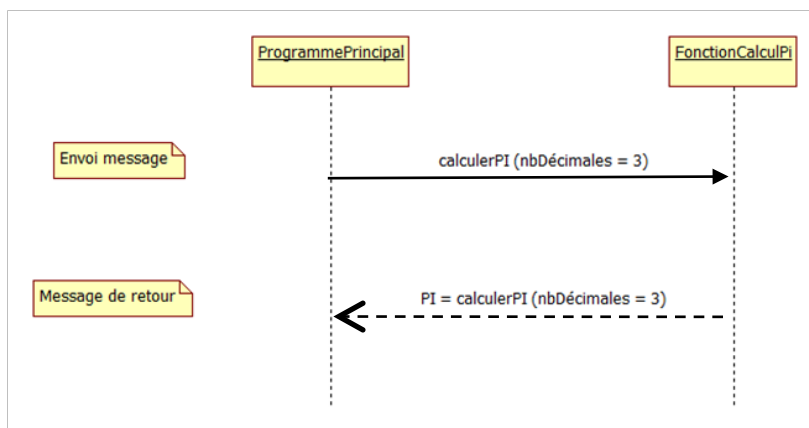
Lorsqu'on envoie un email, on n'attend pas la réponse d'un destinataire. Nous avons alors un message asynchrone.

b) Transmission d'un courrier électronique via un serveur de messagerie qui réceptionne les messages et les conserve en attendant que le destinataire les récupère.



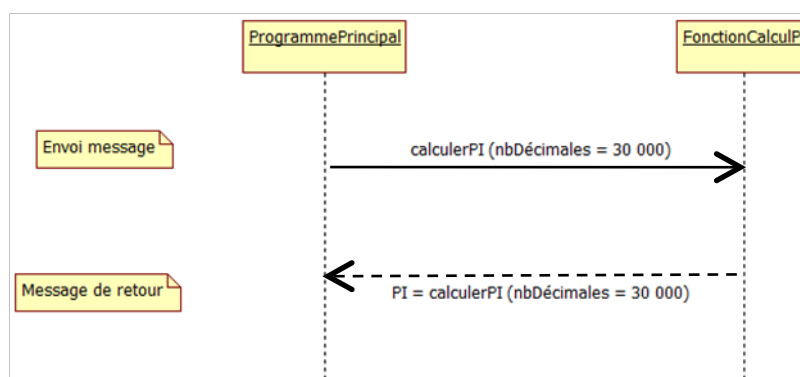
Là, on modélise l'interaction avec le serveur. Il s'agit ici de l'action du protocole SMTP : l'émetteur reste bloqué pendant son dialogue avec le serveur, du même que le destinataire, qui reste bloqué en attendant un message de retour avec ses mails.

c) Envoi d'un message pour calculer la valeur du constant pi avec 3 décimales.



Le calcul du Pi se fait avec une précision réduite, comme en témoigne le paramètre nbDécimales=3. On peut alors se permettre de garder le programme principal bloqué en attendant la réponse de la fonction de calcul du pi. Message synchrone, suivi d'un message de retour.

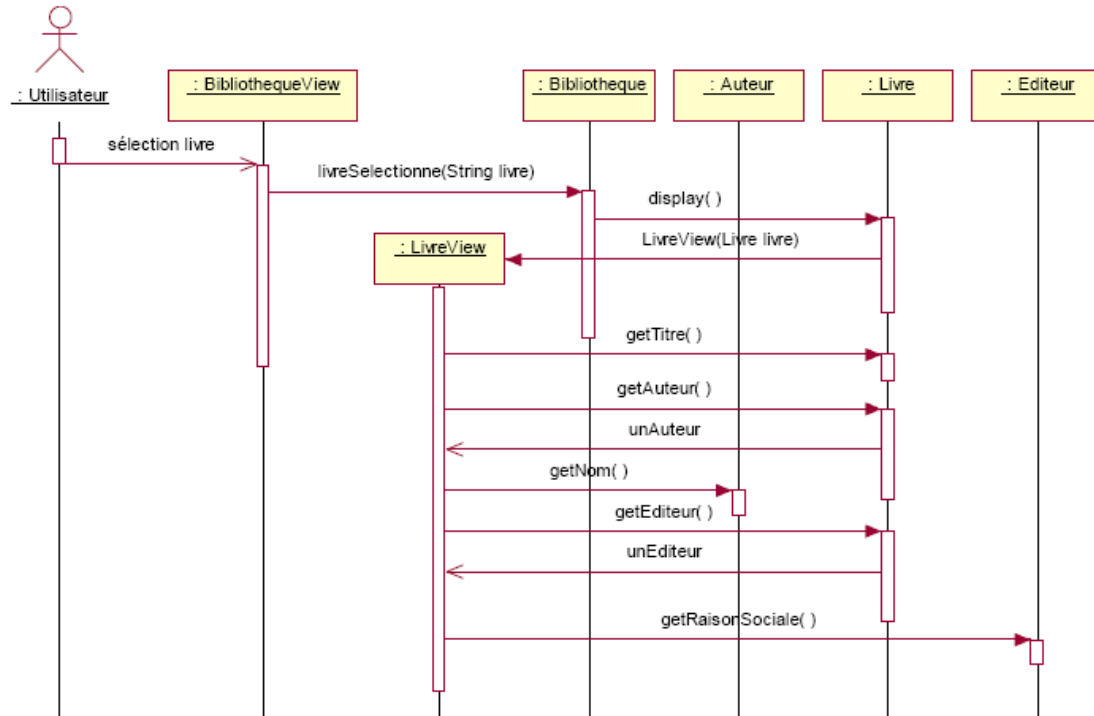
d) Envoi d'un message pour calculer la valeur du constant pi avec 30 000 décimales.



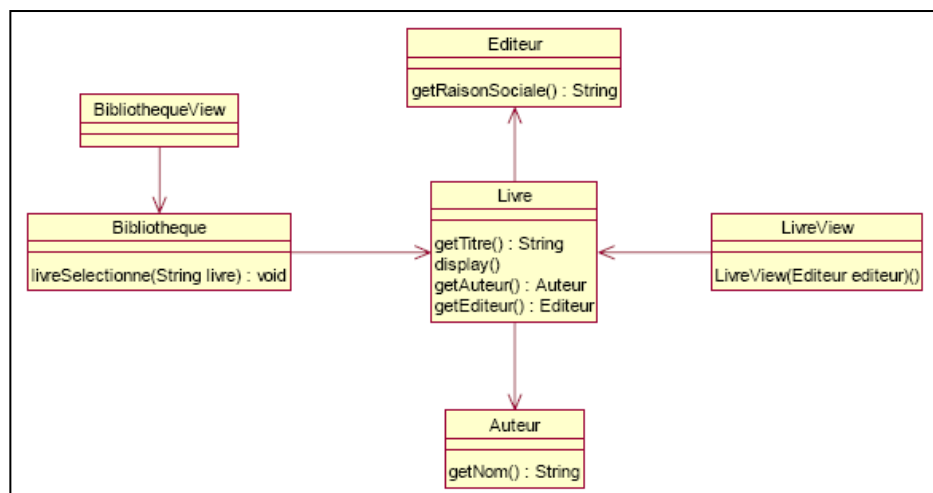
Ici, nous avons un calcul avec une précision importante, indiquée par nbDécimales=30000. Le calcul du Pi dans ces conditions risque de prendre un certain temps. Il n'est donc pas faisable de bloquer le programme principal en attendant la

réponse. On a donc un message asynchrone, suivi d'un message de retour.

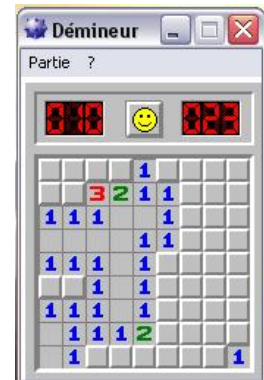
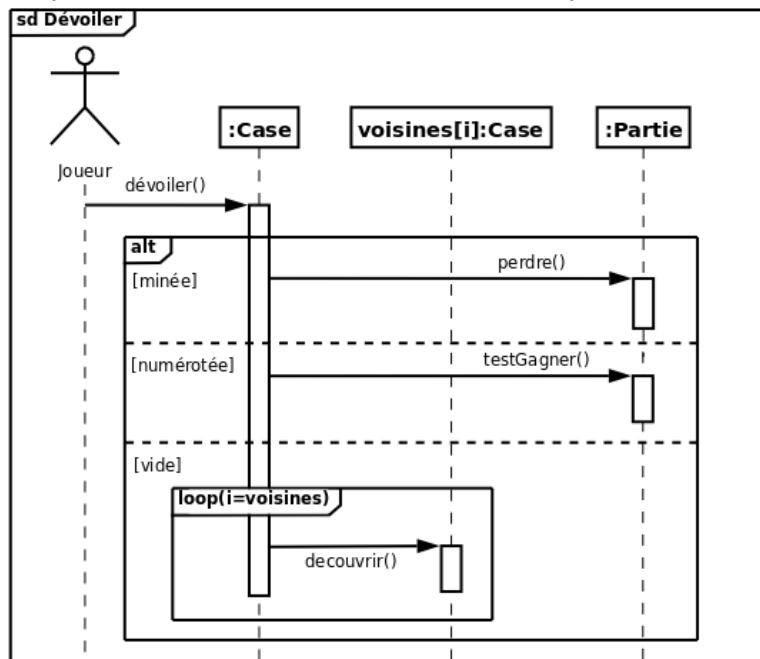
6) Représentez un diagramme de classe compatible avec le diagramme de séquence ci-dessous.



Attention à l'acteur « utilisateur » : en tant qu'acteur, il est extérieur au système, et donc pas présent dans le diagramme de classes. Eventuellement, on peut ajouter une opération « sélectionLivre » dans BibliothèqueView. La navigabilité de l'association entre Livre et LivreView est questionnable, étant donné que Livre doit être capable de créer un LivreView, ce que nous fait penser à une dépendance « create » dans le sens Livre -> LivreView, et une association, telle qu'indiqué dans la solution, dans l'autre sens. A vous de décider si ça vaut la peine d'en parler, selon le niveau du groupe.



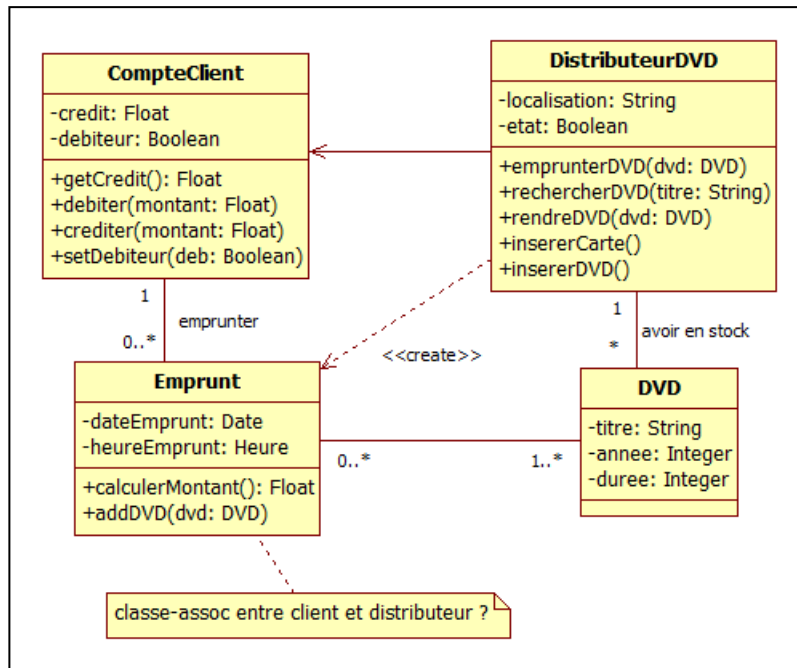
- 7) Modélisez, par un diagramme de séquence, le fonctionnement du jeu démineur. Considérez pour cela les classes « actor » *Joueur*, *Partie*, *Case*, sachant qu'une case a plusieurs cases voisines, lesquelles seront dévoilées si une case vide adjacente est dévoilée.



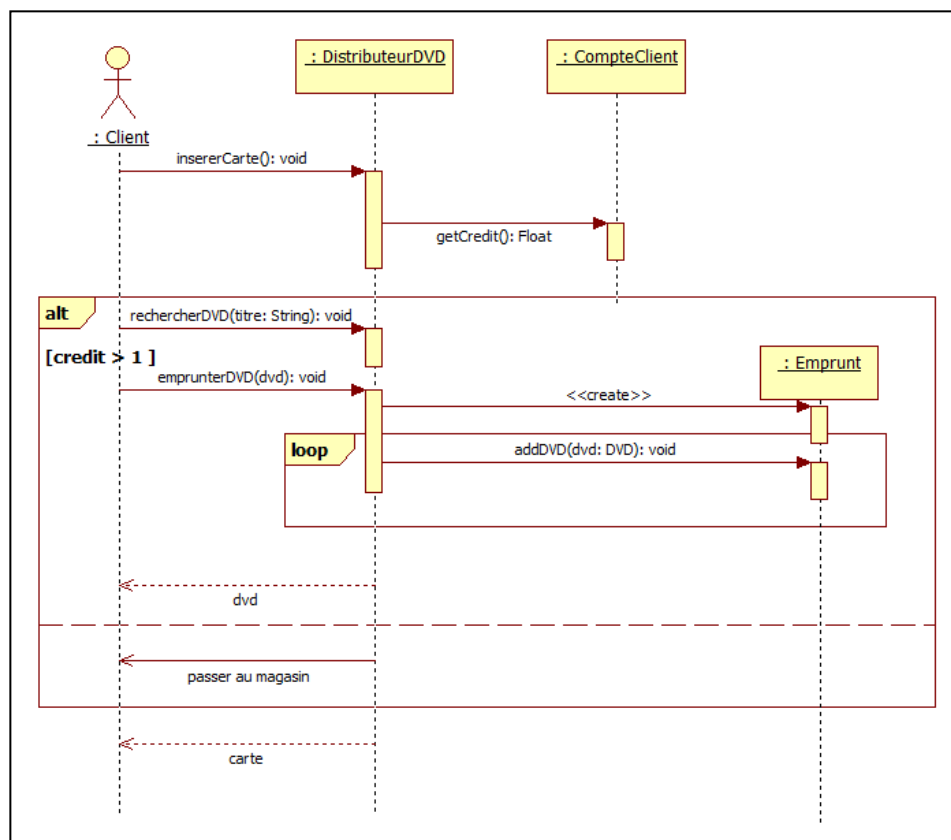
- 8) Le scénario ci-dessous décrit le fonctionnement d'un distributeur automatique de DVD. A partir de ce scénario, modélisez le système de gestion d'emprunts du distributeur :
- Modélisez la structure logique du système en question par un diagramme de classes
 - Modélisez l'action d'emprunter un DVD par un diagramme de séquence
 - Modélisez l'action de rendre un DVD par un second diagramme de séquence

Une personne souhaitant utiliser le distributeur doit avoir une carte magnétique spéciale. Les cartes sont disponibles au magasin qui gère le distributeur. Elles sont créditées d'un certain montant en euros et rechargeables au magasin. Le prix de la location est fixé par tranche de 6 heures (1€ par tranche). Le fonctionnement du distributeur est le suivant : le client introduit sa carte ; si le crédit est supérieur ou égal à 1€, le client est autorisé à louer une cassette (il est invité à aller recharger sa carte au magasin sinon). Le client choisit alors une cassette et part avec. Quand il la ramène, il l'introduit dans le distributeur puis insère sa carte. Celle-ci est alors débitée ; si le montant du débit excède le crédit de la carte, le client est invité à régulariser sa situation au magasin et le système mémorise le fait qu'il est débiteur. On ne s'intéresse ici qu'à la location des DVDs, et non à la gestion du distributeur par le personnel du magasin.

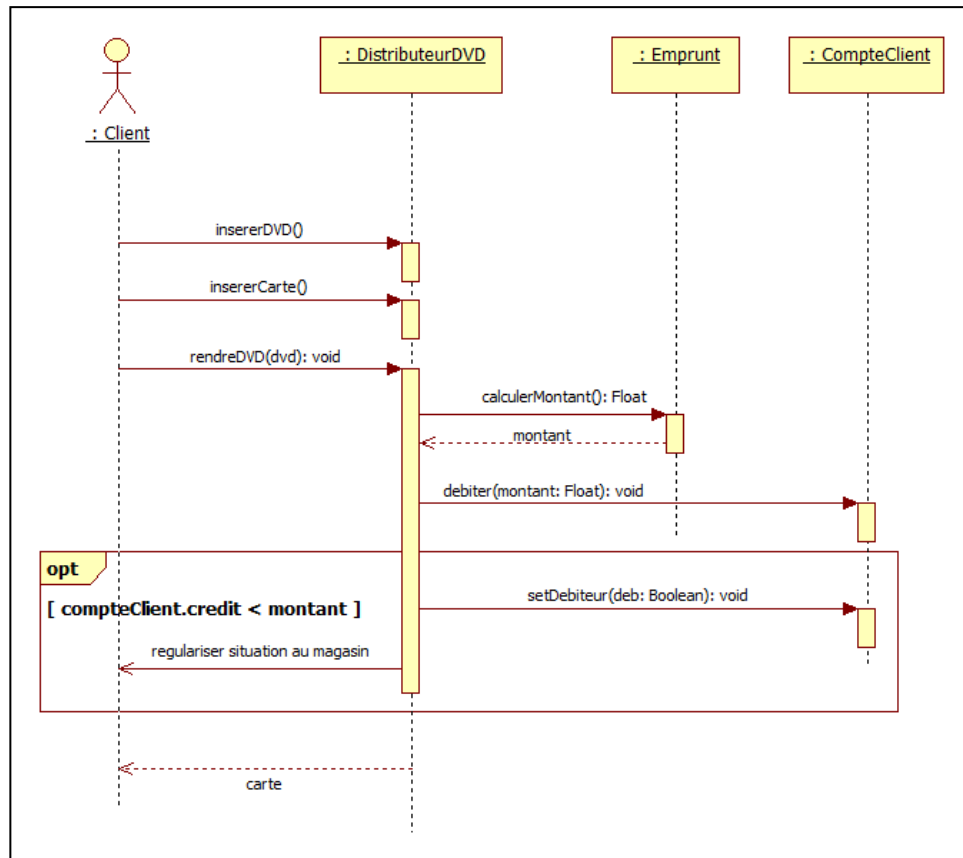
- a) Diagramme de classes



b) processus d'emprunt d'un DVD



d) processus de restitution d'un DVD



- 9) L'étude de cas suivant illustre une application d'agenda électronique qui fait la gestion de l'emploi de temps et des notes.

L'application d'agenda doit permettre à son utilisateur de définir trois types d'éléments d'agenda : une tâche, un rendez-vous, ou encore un contact. Les tâches sont identifiées par leur libellé. Elles peuvent se dérouler sur une certaine plage, avec une date et heure de début et une date et heure de fin. Elles peuvent également compter un commentaire sur leur contenu et une priorité (haute, normal, basse). Chaque tâche peut être associée à une ou plusieurs catégories (famille, professionnel, divers, loisir...). Un rendez-vous comporte un libellé et une plage horaire, avec une date de début, une date de fin, ainsi qu'une heure de début et une heure de fin. En outre, un rendez-vous peut avoir une périodicité et exclure ou non toute autre activité. Il doit être possible de déplacer un rendez-vous et de l'associer aux catégories.

Deux types de contacts sont gérés par l'agenda : les personnes et les organisations. En plus des informations classiques, on aura soin de conserver l'adresse électronique ainsi que le site Internet (celui-ci uniquement pour les organisations). De la même manière que les tâches et les rendez-vous, les contacts sont aussi associés aux catégories.

L'ensemble de l'agenda peut être visualisé, à la demande de l'utilisateur, à travers un calendrier. Trois modes de visualisation sont possibles : à la journée, à la semaine et au mois.

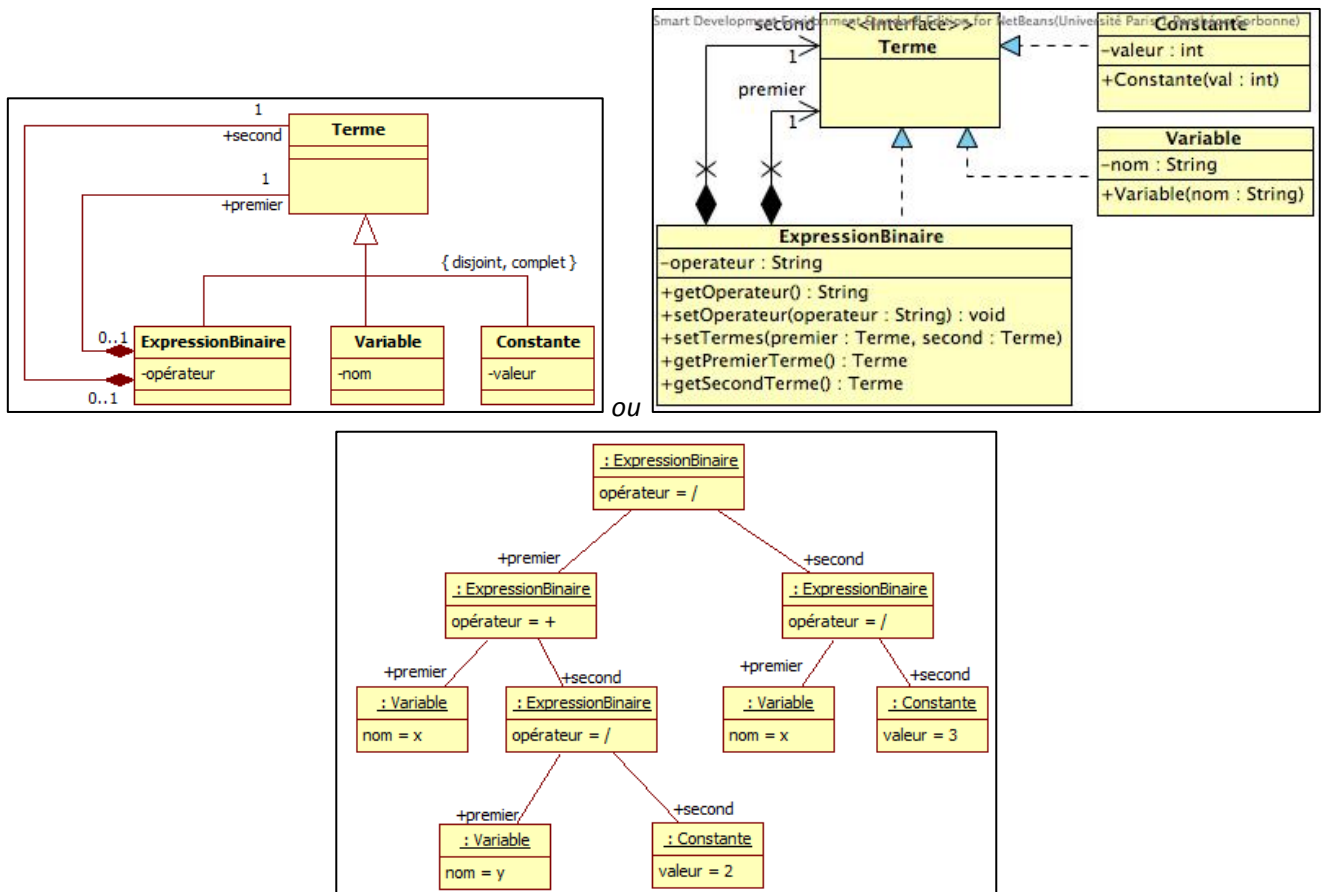
A partir de l'étude de cas précédent, déterminez et modélisez :

- Le diagramme de classes décrivant les données manipulées
- Un diagramme d'objets illustrant le diagramme de classe précédent
- Le diagramme de paquetage, organisant les classes identifiées

Exercices pratiques :

- 1) Le scénario ci-contre représente la notion d'expression binaire. Donner un diagramme de classes permettant de représenter ces expressions et un diagramme objets représentant l'expression $(x+y/2) / (x/3)$. Donner également une implémentation Java correspondant au modèle proposé.

Une expression binaire est constituée de deux termes et d'un opérateur. Par exemple : $x*2$, $x + y/2$, $(x+y/2) / (x/3)$. Un terme est soit une expression binaire, soit une constante, soit une variable. Par exemple : $y/2$, 2, y .



Interface Terme

```
package modelisationreims.expressions;
public interface Terme {
}
```

Classe Variable

```
package modelisationreims.expressions;
public class Variable implements Terme {
    private String nom;
    public Variable(String nom) {
        this.nom = nom;
    }
    public String getNom() {
        return this.nom;
    }
    @Override
    public String toString() {
```

```

        return this.nom;
    }
}

```

Classe Constante

```

package modelisationreims.expressions;
public class Constante implements Terme {
    private int valeur;
    public Constante(int val) {
        this.valeur = val;
    }
    public int getValeur() {
        return this.valeur;
    }
    @Override
    public String toString() {
        return Integer.toString(valeur);
    }
}

```

Classe ExpressionBinaire

```

package modelisationreims.expressions;
public class ExpressionBinaire implements Terme {
    private String operateur;
    public Terme premier;
    public Terme second;
    public String getOperateur() {
        return this.operateur;
    }
    public void setOperateur(String operateur) {
        this.operateur = operateur;
    }
    public void setTermes(Terme premier, Terme second) {
        this.premier = premier;
        this.second = second;
    }
    public Terme getPremierTerme() {
        return this.premier;
    }
    public Terme getSecondTerme() {
        return this.second;
    }
    @Override
    public String toString() {
        return "(" + premier + operateur + second + ")";
    }
}

```

Classe ExpressionMain

```

package modelisationreims.expressions;
public class ExpressionMain {
    public static void main(String[] args) {
        // expression (x+y/2) / (x / 3)

        //expression y/2
        ExpressionBinaire b1 = new ExpressionBinaire();
        b1.setOperateur("/");
        b1.setTermes(new Variable("y"), new Constante(2));

        //expression x+y/2
        ExpressionBinaire b2 = new ExpressionBinaire();
        b2.setOperateur("+");
    }
}

```

```

b2.setTermes(new Variable("x"), b1);

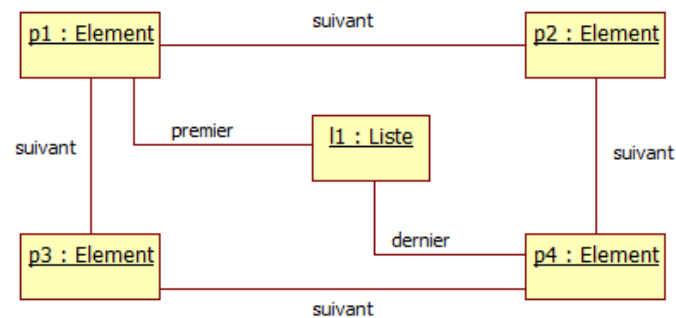
//expression x / 3
ExpressionBinaire b3 = new ExpressionBinaire();
b3.setOperateur("/");
b3.setTermes(new Variable("x"), new Constante(3));

//expression complete
ExpressionBinaire b = new ExpressionBinaire();
b.setOperateur("/");
b.setTermes(b2, b3);

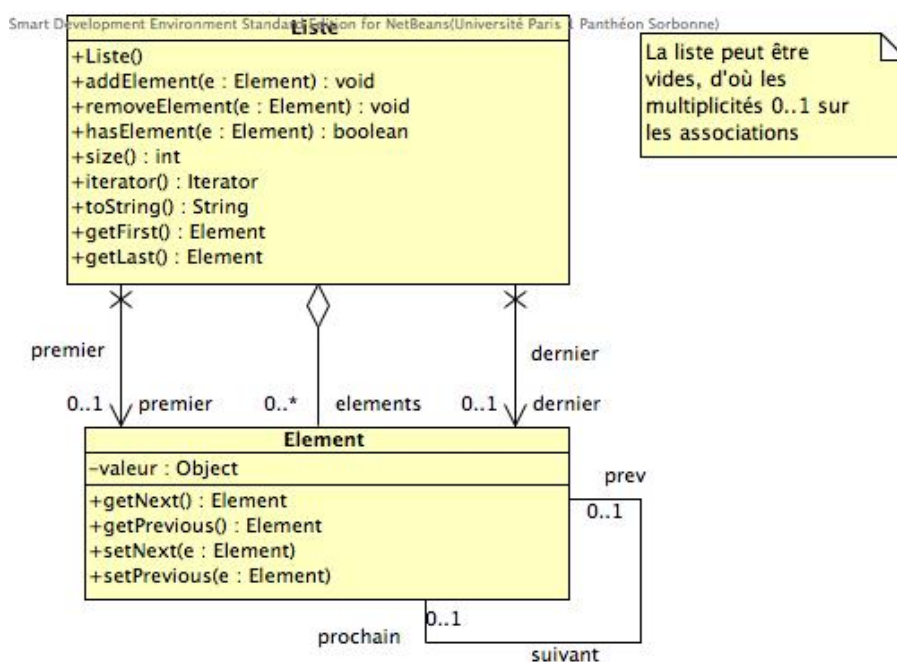
System.out.println(b);
    }
}
    
```

2) A partir du schéma ci-dessous, montrant les instances d'un ensemble de classes, construisez :

- Un diagramme de classes
- Code Java implémentant ces classes



Il s'agit d'une liste circulaire (le suivant du dernier est le premier)



Classe Liste

```

package modelisationreims.liste;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
public class Liste {
    public List<Element> elements;
    public Element premier;
    public Element dernier;
    public Liste() {
        this.elements = new ArrayList<Element>();
        this.premier = null;
        this.dernier = null;
    }
    public void addElement(Element e) {
        //on ajoute l'element
        this.elements.add(e);
        //on l'enchaîne au dernier
        if (this.dernier != null && this.premier!=null) {
            this.dernier.setNext(e);
            e.setPrevious(this.dernier);
            this.dernier = e;
        } else {
            //pas de dernier, donc liste vide, c'est le 1er element de la liste
            this.premier = e;
            this.dernier = e;
        }
        //il s'agit d'une liste circulaire
        //dernier->suivant = premier
        this.dernier.setNext(this.premier);
        this.premier.setPrevious(this.dernier);
    }
    public void removeElement(Element e) {
        //avant de le supprimer, on refait l'enchaînement
        Element apres = e.getNext();
        Element avant = e.getPrevious();
        if (apres != null) {
            apres.setPrevious(avant);
        }
        if (avant != null) {
            avant.setNext(apres);
        }
        //on verifie s'il s'agit du premier ou du dernier
        if (e == this.premier) {
            this.premier = apres;
        }
        if (e == this.dernier) {
            this.dernier = avant;
        }

        //maintenant on peut supprimer l'element
        this.elements.remove(e);
    }
    public boolean hasElement(Element e) {
        return this.elements.contains(e);
    }
    public int size() {
        return this.elements.size();
    }
}
    
```

```

public Iterator<Element> iterator() {
    return this.elements.iterator();
}
@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    Iterator<Element> it = this.elements.iterator();
    sb.append("(");
    while (it.hasNext()) {
        Element e = it.next();
        sb.append(e.getValeur());
        sb.append(",");
    }
    sb.append(")");
    return sb.toString();
}
public Element getFirst() {
    return this.premier;
}
public Element getLast() {
    return this.dernier;
}
}
    
```

Classe Element

```

package modelisationreims.liste;
public class Element {
    private Object valeur;
    public Element prochain;
    public Element prev;
    //un element n'a pas besoin de connaitre sa liste
    //public Liste a_Liste_;
    public Element() {
        this.valeur = null;
        this.prev = null;
        this.prochain = null;
    }
    public Element getNext() {
        return this.prochain;
    }
    public void setNext(Element e) {
        this.prochain = e;
    }
    public Element getPrevious() {
        return this.prev;
    }
    public void setPrevious(Element e) {
        this.prev = e;
    }
    public void setValeur(Object valeur) {
        this.valeur = valeur;
    }
    public Object getValeur() {
        return this.valeur;
    }
    @Override
    public String toString() {
        return this.valeur.toString();
    }
}
    
```


Classe ListeMain

```

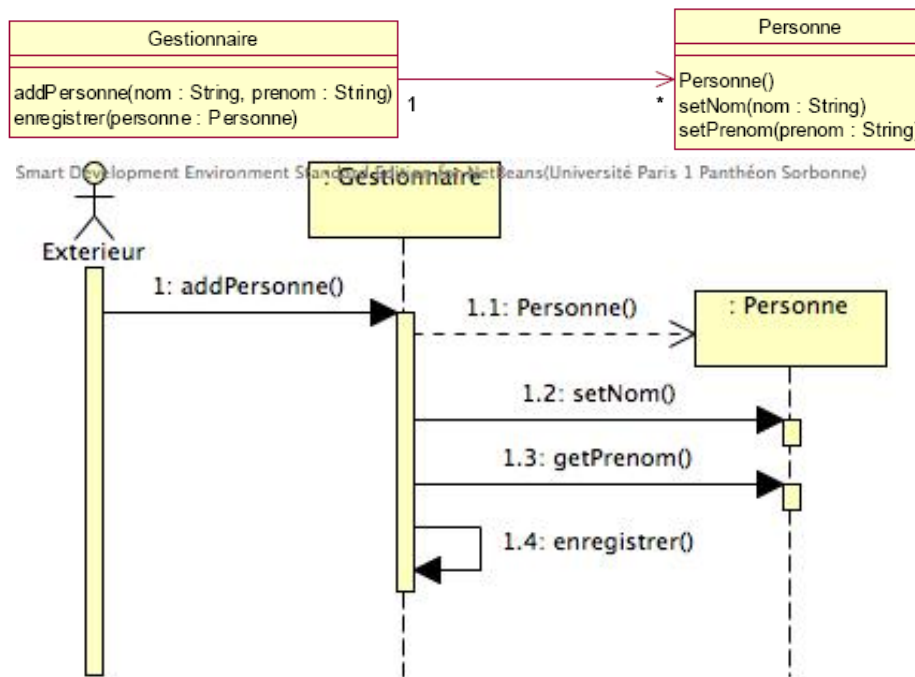
package modelisationreims.liste;
import java.util.Iterator;
public class ListeMain {
    Liste liste = new Liste();
    /**
     * monte une liste d'exemple
     */
    public void createListe() {
        for (int i =0; i<10; i++) {
            Element e = new Element();
            e.setValeur(i);
            liste.addElement(e);
        }
    }
    /**
     * supprime 3 elements de la liste : le premier, le dernier et
     * un au milieu
     */
    public void deleteElements() {
        Element premier = liste.getFirst();
        Element dernier = liste.getLast();
        Element e = null;
        Iterator<Element> it = liste.iterator();
        int count = liste.size();
        count = (count/2)+1;
        for (int i=0;it.hasNext() && i<count; i++) {
            e = it.next();
        }
        liste.removeElement(premier);
        liste.removeElement(dernier);
        liste.removeElement(e);
    }
    /**
     * parcourt la liste, element par element
     */
    public void parcours () {
        Element p = liste.getFirst();
        Element d = liste.getLast();
        Element e = p;
        System.out.println("premier "+p+" dernier "+d);
        //on va du premier au dernier
        while ( (e != null) && (e!=d)) {
            System.out.print(" "+e.toString()+" ");
            e = e.getNext();
        }
        if (e == null)
            System.out.println(" ");
        else
            System.out.println(e.toString());
    }
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        ListeMain m = new ListeMain();
        m.createListe();
        System.out.println(m.liste);
        m.parcours();
    }
}

```

```

        m.deleteElements();
        System.out.println(m.liste);
        m.parcours();
    }
}
    
```

- 3) Créez un diagramme de séquence représentant l'interaction suivante : lorsque le gestionnaire reçoit de la part d'un acteur extérieur le message *addPersonne*, il va créer une nouvelle instance de la classe *Personne*, lui assigner le nom et le prénom, puis il va enregistrer la nouvelle personne à l'aide de l'opération *enregistrer*. Créez ensuite le code Java implémentant cette interaction.



Classe Personne

```

package modelisationreims.gestionpersonne;
public class Personne {
    private String nom;
    private String prenom;
    public Personne() {
        this.nom = "";
        this.prenom = "";
    }
    public void setNom(String nom) {
        this.nom = nom;
    }
    public void setPrenom(String prenom) {
        this.prenom = prenom;
    }
    public String getNom() {
        return this.nom;
    }
    public String getPrenom() {
        return this.prenom;
    }
}
    
```

Classe Gestionnaire

```
package modelisationreims.gestionpersonne;
import java.util.ArrayList;
import java.util.Iterator;
public class Gestionnaire {
    public ArrayList<Personne> a_Personne_ = new
ArrayList<Personne>();
    public void addPersonne(String nom, String prenom) {
        Personne p = new Personne();
        p.setNom(nom);
        p.setPrenom(prenom);
        this.a_Personne_.add(p);
        this.enregistrer(p);
    }
    public void enregistrer(Personne personne) {
        //on pourrait enregistrer sur un fichier
        //en attendant, on affiche a l'ecran
        System.out.println(personne.getNom()+"",
"+personne.getPrenom());
    }
}
```

Classe GestionnaireMain

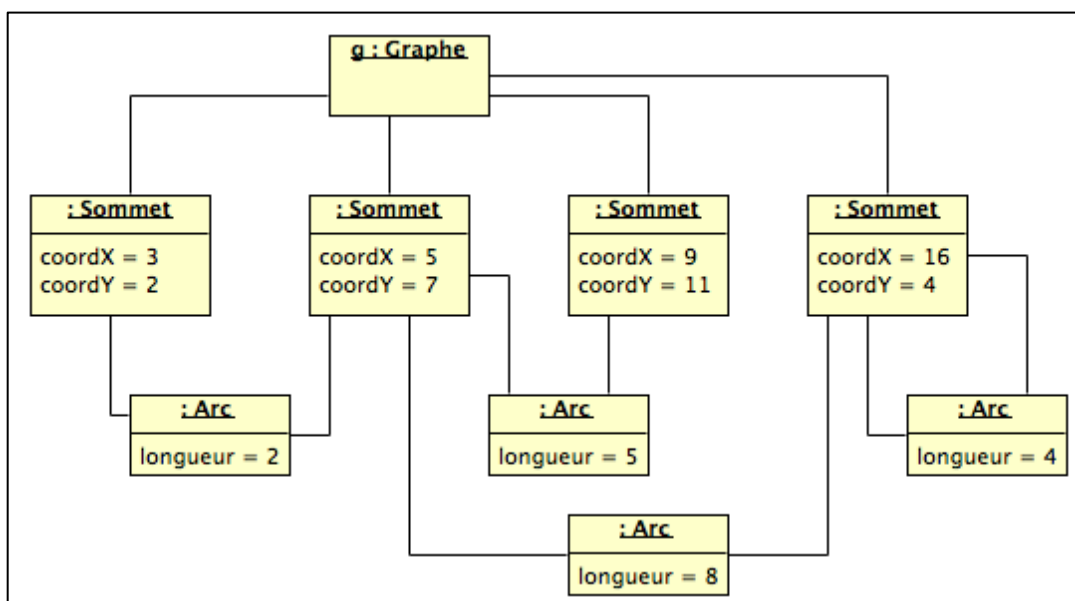
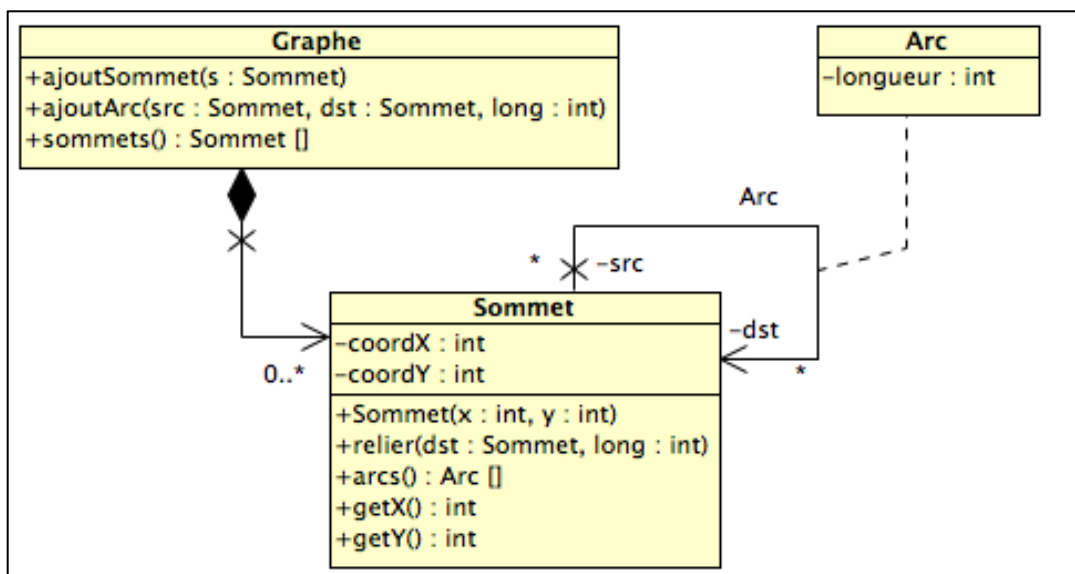
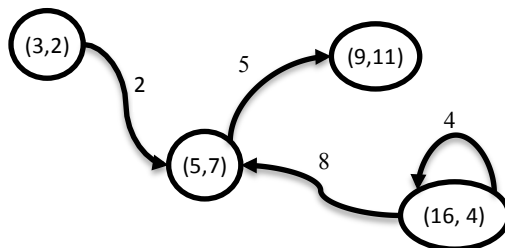
```
package modelisationreims.gestionpersonne;
import java.util.Scanner;
public class GestionnaireMain {
    Gestionnaire g = new Gestionnaire();

    public void lecturePersonne() {
        String nom;
        String prenom;
        //on lit les noms et les prenom du clavier
        Scanner scan = new Scanner (System.in);

        System.out.print("nom ?");
        nom = scan.nextLine();
        System.out.print("prenom ?");
        prenom = scan.nextLine();

        g.addPersonne(nom, prenom);
    }
    public static void main(String[] args) {
        GestionnaireMain m = new GestionnaireMain();
        m.lecturePersonne();
        m.lecturePersonne();
    }
}
```

- 4) La figure ci-contre illustre un graphe dirigé. Un graphe dirigé est un ensemble de sommets et d'arcs orientés. Les arcs relient deux sommets (une source et un destinataire). Chaque sommet est un point dans un plan. Il a donc des coordonnées X et Y. Les arcs reliant les sommets peuvent avoir alors une longueur. Modéliser la notion de graphe par un diagramme de classes. Implémenter le modèle correspondant en Java.



Attention : UML ne propose pas une représentation claire pour les classe-associations au niveau du diagramme objets. Par contre, on sait qu'un objet d'une telle classe n'apparaît que si le lien entre les objets existe. On peut donc le représenter comme intermédiaire entre les deux objets.

Classe Arc

```
package modelisationreims.graphe;

public class Arc {
    private int longueur;
    private Sommet src;
    private Sommet dst;

    public void setSrc(Sommet src) {
        this.src = src;
    }

    public Sommet getSrc() {
        return this.src;
    }

    public void setDst(Sommet dst) {
        this.dst = dst;
    }

    public Sommet getDst() {
        return this.dst;
    }

    public void setLongueur(int longueur) {
        this.longueur = longueur;
    }

    public int getLongueur() {
        return this.longueur;
    }
}
```

Classe Sommet

```
package modelisationreims.graphe;

import java.util.ArrayList;
import modelisationreims.graphe.Arc;

public class Sommet {

    private int coordX;
    private int coordY;
    private ArrayList<Arc> a_Arc_ = new ArrayList<Arc>();

    public Sommet(int x, int y) {
        this.coordX = x;
        this.coordY = y;
    }

    public void relier(Sommet dst, int long_1) {
        Arc a = new Arc();
        a.setLongueur(long_1);
    }
}
```

```

        a.setSrc(this);
        a.setDst(dst);
        this.a_Arc_.add(a);
    }

    public Arc[] arcs() {
        return this.a_Arc_.toArray(new Arc[0]);
    }

    public int getX() {
        return this.coordX;
    }

    public int getY() {
        return this.coordY;
    }
}

```

Classe Graphe

```

package modelisationreims.graphe;
import java.util.ArrayList;
import modelisationreims.graphe.Sommet;

public class Graphe {

    private ArrayList<Sommet> a_Sommet_ = new ArrayList<Sommet>();

    /**
     * ajoute un sommet au graphe
     * @param s nouveau sommet
     */
    public void ajoutSommet(Sommet s) {
        this.a_Sommet_.add(s);
    }

    /**
     * on ajoute un arc entre les sommets src et dst. Si les sommets
     * ne sont pas d'jà dans le graphe, on les ajoute.
     * @param src sommet source
     * @param dst sommet destination
     * @param long_22 longueur de l'arc
     */
    public void ajoutArc(Sommet src, Sommet dst, int long_22) {
        if (!this.a_Sommet_.contains(src)) {
            this.a_Sommet_.add(src);
        }
        if (!this.a_Sommet_.contains(dst)) {
            this.a_Sommet_.add(dst);
        }
        src.relier(dst, long_22);
    }

    /**
     * retourne un array avec les sommets qui composent le graphe
     * @return sommets du graphe
     */
    public Sommet[] sommets() {
        return this.a_Sommet_.toArray(new Sommet[0]);
    }
}

```

Classe MainGraphe

```

package modelisationreims.graphe;

public class MainGraphe {
    Graphe g = new Graphe();

    public void creerGraphe() {
        Sommet src;
        Sommet dst;

        //(3,2), (5,7) arc longueur 2
        src = new Sommet(3,2);
        dst = new Sommet(5,7);
        g.ajoutArc(src, dst, 2);

        //(16,4) (5,7) arc longueur 8
        src = new Sommet(16,4);
        g.ajoutArc(src, dst, 8);

        //(16,4) (16,4) arc longueur 4
        g.ajoutArc(src, src, 4);

        //(5,7) (9,11) arc longueur 5
        src = dst; //(5,7) devient src
        dst = new Sommet(9,11);
        g.ajoutArc(src, dst, 5);
    }

    public String printSommet (Sommet s) {
        return "("+s.getX()+","+s.getY()+") ";
    }

    public String printArc (Arc a) {
        return ("\t - "+a.getLongueur()+" - "
            + printSommet(a.getDst()));
    }

    public void parcourirGraphe() {
        Sommet[] sommets = g.sommets();

        for (Sommet s: sommets) {
            System.out.println(printSommet(s));
            for (Arc a: s.arcs()) {
                System.out.println(printArc(a));
            }
        }
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        MainGraphe m = new MainGraphe();
        m.creerGraphe();
        m.parcourirGraphe();
    }
}

```

Projet¹

Répondez aux questions ci-dessous à partir du scénario indiqué :

*Vous êtes engagés par une entreprise afin de développer une solution client/serveur pour leur système de réservation en ligne de billets (places de cinéma, de spectacles, de concerts, etc.). Chaque billet est identifié par un numéro unique et un statut (disponible, bloqué, vendu), et selon le type de billet des informations supplémentaires doivent être incluses (**cinéma** : film, réalisateur, année, langue, sous-titrage, 3D, limitation d'âge, salle d'exhibition, date, heure ; **théâtre** : pièce, auteur, metteur en scène, date, heure, catégorie du siège ; **sport** : nom de l'événement, date, heure, catégorie du billet ; etc.).*

Les étapes de réservation sont les suivantes :

- 1. le client demande une liste d'événements au serveur ;*
- 2. le serveur répond avec une liste d'événements (ordonnée par nom) ;*
- 3. le client demande au serveur une place disponible sur un événement spécifique (nom, date et heure) ;*
- 4. le serveur répond avec le numéro d'une place disponible (cette place est alors bloquée pendant un temps t)*
- 5. trois actions sont possibles :*
 - a. le client envoie au serveur le numéro de la place qu'il souhaite ;*
 - b. le client indique au serveur qu'il souhaite annuler la commande ;*
 - c. le temps t est expiré et la commande du billet est annulée ;*
- 6. Si le message reçu est une confirmation, le serveur valide la transaction (modifie le statut de la place vers VENDU) et enregistre les données du client. Il envoie ensuite le billet au client. De même, le serveur envoie un message au fournisseur du service (cinéma, théâtre) afin de lui informer de la vente du billet.*

Étapes du développement :

- 1) Modéliser les données mentionnées dans le scénario ci-dessus par un diagramme de classes.
- 2) Modéliser le système de réservation par un (ou plusieurs) diagrammes de classes.
- 3) Modéliser le processus de réservation d'un nouveau billet par un diagramme de séquence.
- 4) Implémenter en Java le système proposé.

¹ Projet en collaboration avec le cours « Programmation Client-Serveur ».