Modélisation et programmation par objets 2 - Correction

1 Interface, classe générique, streams, assertion (environ 2/3 des points)

Dans cette partie, nous développons des éléments pour la représentation de halles thématiques dans un parc exposition. Une halle thématique comprendra plusieurs stands qui partagent une même thématique liée à l'exposition.

Question 1. (2,5 points) Ecrivez en Java une interface représentant un stand dans le contexte d'une halle d'un parc d'exposition. Pour un stand, on peut connaître (1) un descriptif, (2) une durée (en heures), (3) une surface (en m^2), (4) un prix de location de base par heure et par m^2, qui est le même pour tous les stands de l'exposition, est supposé constant et de visibilité `public` (5) un prix de location calculé comme le produit de la surface par la durée.

On acceptera les solutions dans lesquelles il manque `"*Istand.prixLocBaseHM2"`, à cause de la forme de l'énoncé qui laisse ouverte cette interprétation.

```
public interface Istand { // 0,5pt

    String descriptif(); // 0,5pt pour les 3 méthodes descriptif, durée, surface

    int duree();

    double surface();

    double prixLocBaseHM2 = 15; // 0,5pt (seulement 0,25 si méthode juste abstract, ou juste static)

    default double prixLoc(){  // 0,5pt (seulement 0,25 si méthode juste abstract)
        return this.surface()*this.duree()*Istand.prixLocBaseHM2; // 0,5pt
    }

}
```

Question 2. (2 points + 0,25 voir ci-dessous) Ecrivez en Java une classe concrète représentant un stand et implémentant l'interface de la question précédente. N'écrivez que les attributs et que les méthodes suffisant à rendre la classe concrète. N'écrivez pas les constructeurs mais ils seront supposés exister, de même que les accesseurs non écrits.

```
public class Stand implements Istand { // 1pt

    private String descriptif = "inconnu"; // 0,5 pt pour les 3 attributs

    private int duree;

    private double surface;

    public String descriptif() {return this.descriptif;} // 0,5 pt pour les 3 méthodes

    public int duree() {return this.duree;}

    public double surface() {return this.surface;}

    // ajouter 0,25 pt si la méthode prixLoc a son corps écrit ici
}
```

Question 3. (1,5 pt) Ecrivez en Java une assertion vérifiant que le prix de location calculé est bien positif ou nul. Expliquez et indiquez avec précision à quel endroit vous mettez cette assertion.

Comme il s'agit d'une post-condition de la méthode `prixLoc`, cette assertion est placée à la fin de la méthode, mais avant `return` sinon elle ne serait jamais exécutée. // 0,5 pt

```
public interface Istand {
(...)
    default double prixLoc(){
        double pl = this.surface()*this.duree()*Istand.prixLocBaseHM2;
        assert pl >=0; // 1 pt
        return pl;
    }
}
// on peut accepter aussi d'autres réponses comme de vérifier dans toute méthode
// si l'appel de prixLoc() continue à retourner un résultat positif
```

Question 4. (2 pts) Une halle thématique possède un nom, une surface et une liste de stands. Ecrivez en Java l'entête et les attributs (avec initialisation) d'une classe générique représentant une halle thématique. La classe générique est paramétrée par un type de stands (conformes à l'interface définie plus haut).

```
public class HalleThematique<TS extends Istand> { // 0,5 pt pour TS, 0,5 pt pour extends IStand

    private String nom = "inconnu"; // 0,25 pt pour les attributs

    private double surface = 0;

    private ArrayList<TS> listeStands // 0,5 pt
            = new ArrayList<>();   // 0,25 pt
}
```

Question 5. (2,5 pts) Ecrivez en Java, dans la classe représentant les ~~parcs exposition~~ halles thématiques, une méthode `prixTotalLocationStandSurfaceSupÀ` retournant la somme des prix de location des stands dont la surface est supérieure à une certaine surface passée en paramètre. Cette méthode doit être impérativement écrite en utilisant les **opérations sur les flots (streams) et les lambdas expressions** et non pas une classique itération avec un `for` ou un `while`, ni avec un itérateur.

On acceptera les solutions dans lesquelles une classe `Parc Exposition` a été ajoutée, avec une liste de halles et les méthodes permettant de calculer la somme des prix de location des halles et du parc.

Ne pas enlever des points pour les usages inutiles ou oublis de mapToInt, mapToDouble, getAsDouble, ...

```
   double prixTotalLocationStandsSurfaceSupA(double s){ // 0,25 pt pour cette ligne et le return

      return

      listeStands // 0,25 pt

      .stream() // 0,5 pt

      .filter(st -> st.surface()>=s) // 0,5 pt

      .mapToDouble(st -> st.prixLoc()) // 0,5 pt (ou mapToDouble(TS::prixLoc)

      .sum(); // 0,5 pt
   }
```

Question 6. (3 pts) Dessiner en UML un diagramme de classes correspondant aux éléments écrits en Java dans les questions 1 à 4.

Eléments de notation

- (0,5 pts) Interface, son attribut initialisé, ses méthodes abstraites et sa méthode `default`. `default` peut être mis sous forme d'un commentaire UML plutôt que sous forme d'un stéréotype.
- (0,5 pts) L'assertion apparaît comme un commentaire. Si OCL était connu, ce pourrait être un invariant (ce sera étudié en Master).
- (0,5 pts) Classe `Stand`, ses attributs, ses méthodes (les commentaires peuvent ne pas être mis car ils sont très simples ici).
- (0,5 pts) Classe `HalleThematique` (et éventuellement `ParcExposition`) avec un paramètre de généricité (avec ou sans sa borne)
- (0,5 pts) relation `realizes` (le mot-clef n'est pas obligatoire, le trait pointillé et la flèche suffisent) - mettre 0,25 pt si c'est une flèche extends classique
- (0,5 pts) attribut de type liste ou relation de composition entre halle et Istand avec sa cardinalité (avec des variantes possibles). Un stand peut être ou non dans une halle et une halle peut contenir aucun ou plusieurs stands.

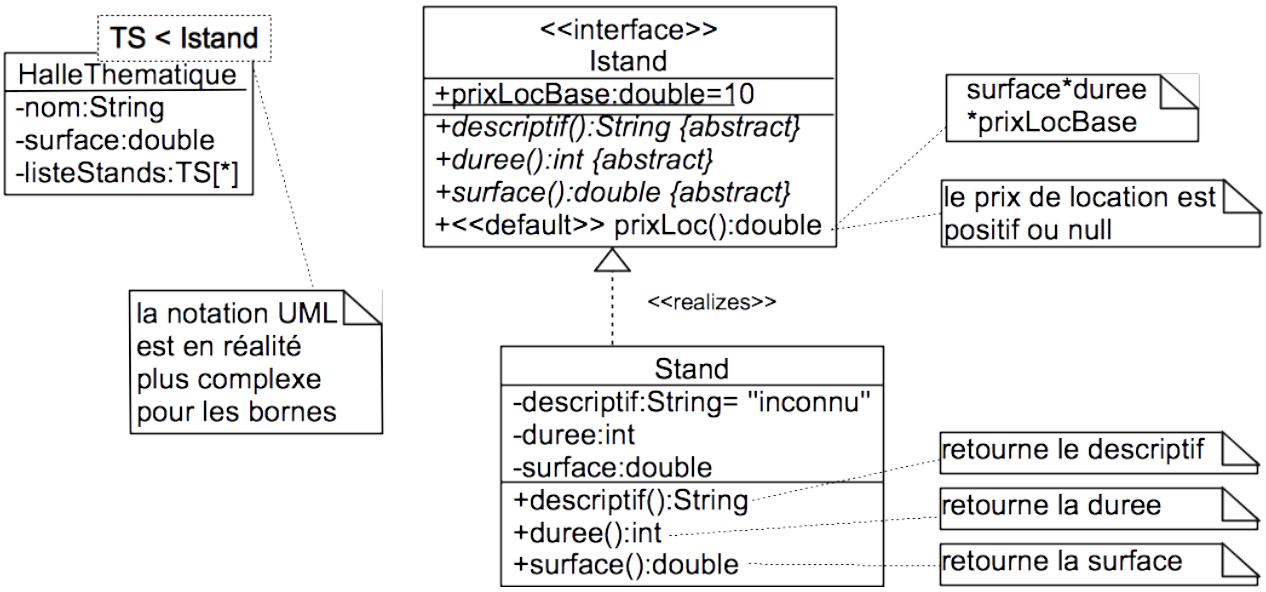

FIGURE 1 – Diagramme de classes pour les questions 1 à 4.

2 Diagramme statique et diagramme dynamique (environ 1/3 des points)

Question 7. (7 pts) Vous représenterez en UML la situation suivante par (1) **un diagramme de classes** et (2) **un diagramme de séquences** cohérents l'un avec l'autre. Nous suggérons de commencer par réaliser le diagramme de séquences pour identifier les opérations nécessaires sur les différentes classes. Nous rappelons que le diagramme de séquence représente des échanges de messages entre instances, le long d'une ligne de vie ; des méthodes doivent être prévues dans les classes pour étiqueter ces messages.

On s'intéresse, dans le domaine bancaire, à la description (ici simplifiée) d'une procédure de virement en ligne par un client sur le compte d'un autre client.

Le premier `client` envoie une requête de virement par l'interface en ligne de sa `banque-locale`, cette requête comprend la somme à virer et le numéro de `compte` du second client.

Si le solde du compte du premier client ne permet pas ce virement, un message d'erreur lui est retourné et l'opération s'arrête.

Sinon la banque du premier client envoie une requête à la `banque-centrale` pour vérifier que le numéro de compte du second client existe et récupérer sa référence.

Si le numéro de compte du second client n'existe pas, un message d'erreur est retourné par la banque centrale à la banque locale qui le fait suivre au premier client et l'opération s'arrête.

Sinon, la référence du compte du second client est transmise à la banque locale et la banque locale effectue le virement : elle retire la somme du compte du premier client et l'ajoute au compte du second client.

Le premier client et la banque locale du second client reçoivent une notification de la part de la banque locale du premier client. La banque locale du second client fait suivre ce message à ce dernier.

Il existe différentes interprétations et modélisations de ce texte. La solution est donnée à titre indicatif. Quelques points de variantes ou de discussion :

- un client n'a qu'un compte (sinon lors du virement le premier client devrait indiquer depuis quel compte il fait le virement).
- les deux clients peuvent être dans la même banque locale.
- on peut décider que banque locale et banque centrale sont des rôles d'une unique classe `Banque` et non des types.
- on peut ajouter des messages entre les banques pour éviter que, comme le stipule l'énoncé, la banque du client 1 dépose directement la somme sur le compte du client 2 sans passer par la banque du client 2.
- on ne met pas forcément les boîtes d'activation des objets sur les lignes de vie.

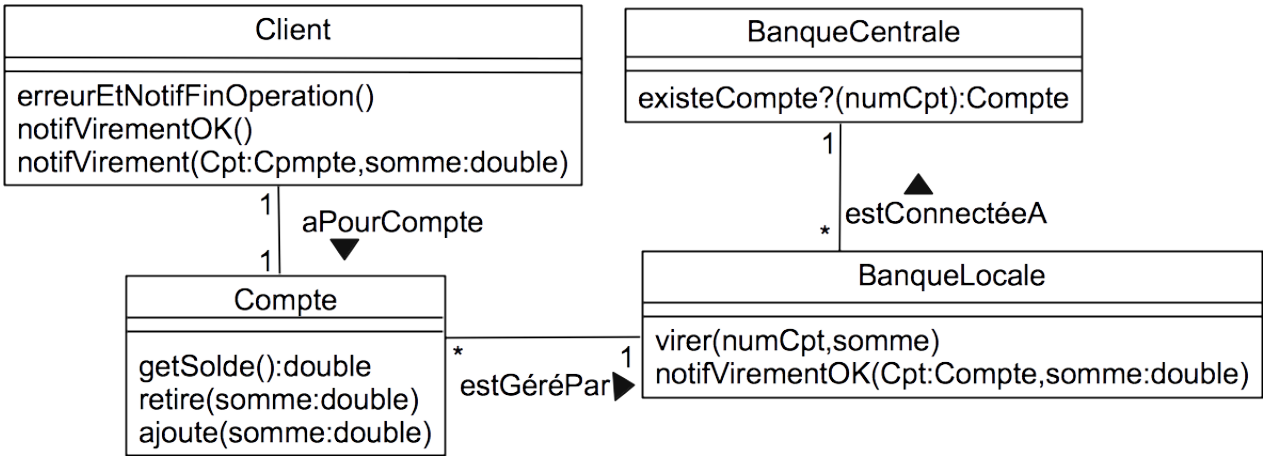

Figure 2 – Diagramme de classes pour la question 7.

Les principaux éléments de notation :

— choisir les deux bons types de diagrammes
— diagramme de séquences :
  — (1 pt) avoir un ensemble d'objets cohérents en tête des lignes de vie pour les clients, les comptes, les (la) banque, avec la bonne syntaxe graphique (' :', type, souligné).
  — (0,5 pt)syntaxe correcte des flèches (appel de méthode synchrone et retour)
  — (1 pt) étiquettes correctes et informatives sur les flèches (noms de messages et paramètres, valeur retournée)
  — (0,5 pt) utiliser deux alternatives avec la syntaxe des fragments (alt et zones).
  — (1 pt) écrire des gardes correctes entre '[....]'
  — (1 pt) respecter la suite logique des opérations (comme indiqué dans le texte).
— diagramme de classes :
  — (1 pt) classes Client, Compte, Banque (éventuellement deux sortes de banque et une super-classe Banque).
  — (1 pt) les méthodes sont dans les bonnes classes.

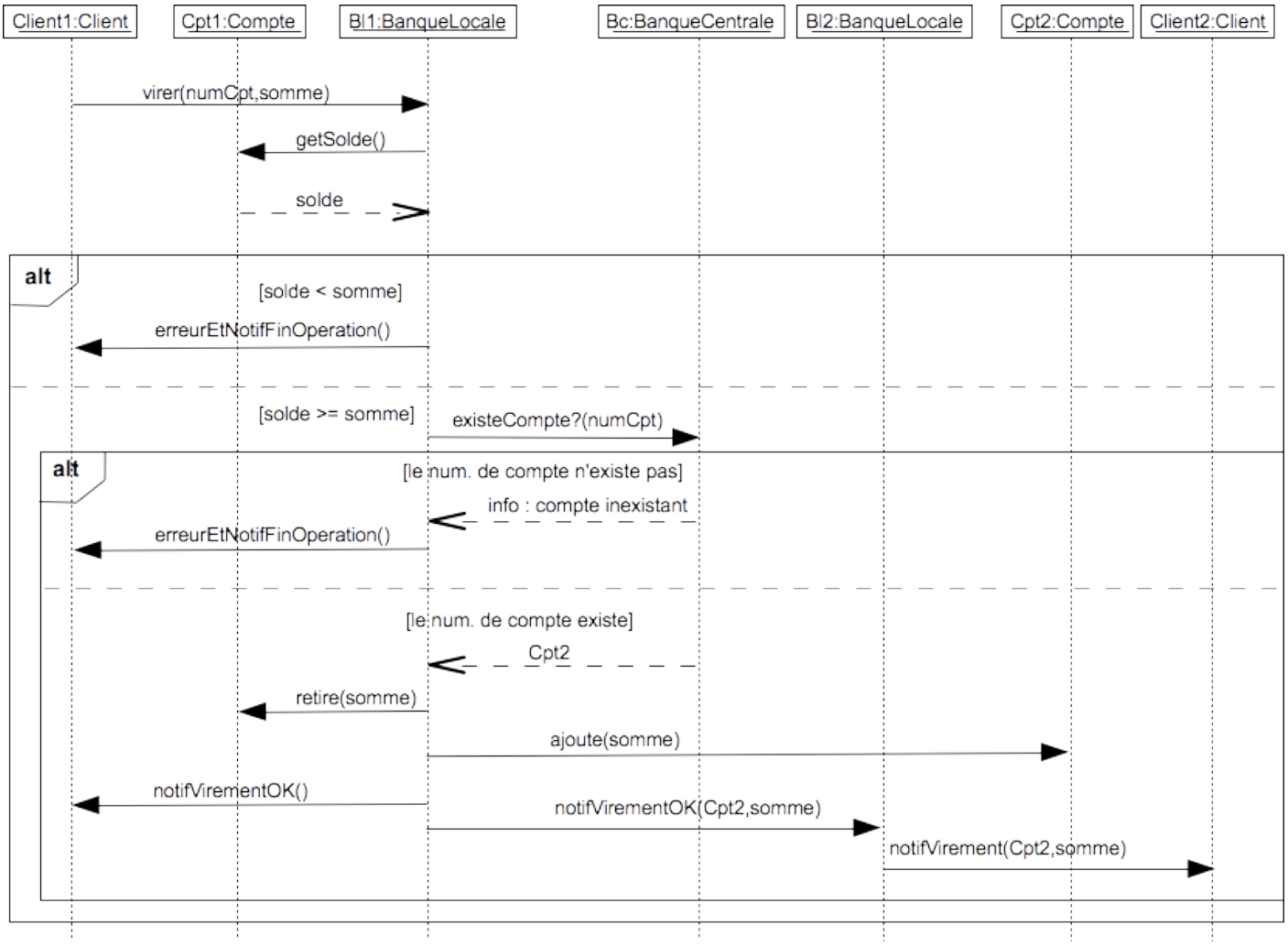

FIGURE 3 – Diagramme de séquences pour la question 7.