1 Présentation

Dans ce sujet, nous présentons quelques éléments simplifiés d'un logiciel pour la gestion des salons qui se tiennent dans un parc d'expositions. Un salon regroupe un ensemble de stands occupant des espaces du parc d'expositions. La partie du logiciel que nous étudierons est destinée à effectuer un suivi des stands. Des noms vous sont proposés pour les classes, les attributs et les opérations que nous vous invitons à respecter.

2 Représentation des espaces et des stands ; première représentation des salons

Un salon est décrit par :
— un nom (`nom`),
— un tarif d'entrée (`tarifEntree`),
— une durée en jours (`dureeEnJours`),
— un nombre d'heures d'ouverture par jour (`nbHeuresOuvertureParJour`),
— un coût de logistique (`coutLogistique`),
— une liste de stands.
Un espace (`Espace`) est décrit par :
— une surface en m^2 (`surfaceEnM2`),
— le stand qui lui est associé (`standAssocie`),
— une indication sur le fait que le stand se trouve en intérieur ou en extérieur (`estInterieur`).
Un stand (`Stand`) est décrit par :
— une identification (`identification`),
— une description (`description`),
— si une connexion à un réseau d'eau est nécessaire (`prestationEauDemandee`),
— le prix de la prestation en eau (`prixPrestationEau`) ; ce prix est le même pour tous les stands, vaut 10 euros au départ et peut évoluer,
— le prix de la location d'un stand par m^2 et par heure (`prixBaseLocationParHeureParM2`) ; ce prix est le même pour tous les stands et vaut 15 au départ ; il peut évoluer,
— l'espace qui lui est associé.
— à quel salon ils appartiennent.
Deux types de stands sont proposés :
— les stands "atelier" (`StandAtelier`), d'une durée courte (quelques heures) et qui accueillent un nombre limité de participants. Un exposant peut y faire une démonstration de ses procédés de fabrication, proposer une lecture, faire déguster ses spécialités, etc.
— les stands "permanents" (`StandPermanent`), dont la durée est celle du salon.
Pour être plus précis, les stands atelier sont des stands avec ces particularités :
— une durée en heures (`dureeAtelierEnHeures`),
— un nombre de participants attendus (`nbParticipantsAttendus`).

Question 1. Ecrivez en Java les classes représentant les stands, les salons, les espaces, munies uniquement de leurs attributs (en les initialisant si ce sont des String ou des ArrayList) et des constructeurs avec paramètres. Il ne sera pas possible de créer d'instances directes de la classe `Stand`. Pour la suite, vous supposerez que les constructeurs sans paramètres et les accesseurs existent pour tous les attributs de ces trois classes. Reportez les attributs dans le schéma UML.

Barême (7 points, dont 1 pour le modèle UML)
- **1pt pour entêtes de classes dont les extends**
- **1 pt pour les attributs en général : type primitif, type classe, static**
- **1 pt pour la ArrayList avec une initialisation à un endroit**
- **1 pt pour les constructeurs simples**
- **1 pt pour les constructeurs avec appel au super constructeur**
- **1 pt pour mettre "abstract"**
- **1 pt pour le modèle UML complété en général (pour toutes les questions)**

```
Voir pages manuscrites fichier Q1.pdf (Moodle)
```

Question 2. Ecrivez en Java les méthodes nécessaires au calcul de la durée en heures de la location d'un stand (`dureeLocationEnHeures`) avec les informations suivantes :
- On suppose qu'un salon est associé au stand.
- Elles retournent une durée (elles ne l'affichent pas).
- Pour un stand atelier, la durée en heures de la location d'un stand est simplement la valeur de l'attribut `dureeAtelierEnHeures`.
- Pour un stand permanent, cette durée est donnée par le produit :
  - du nombre de la durée d'ouverture du salon en jours
  - et du nombre d'heures d'ouverture du salon par jour.

Reportez la signature (entête) des méthodes dans le schéma UML.

Barême (3 pts)
- **1pt pour méthode abstraite dans Stand**
- **1 pt pour méthode redéfinie dans StandAtelier**
- **1 pt pour méthode redéfinie dans StandPermanent**

```
// Dans Stand
public abstract int dureeLocationEnHeures();

// Dans StandAtelier
public int dureeLocationEnHeures() {return this.dureeAtelierEnHeures;}

// Dans StandPermanent
public int dureeLocationEnHeures() {
   return this.getSalonAssocie().getDureeEnJours()
             *this.getSalonAssocie().getNbHeuresOuvertureParJour();
}
```

Question 3. On ajoute dans la classe `Stand` une méthode `toString` écrite de la manière suivante (dans un style qui n'est pas dans l'esprit de l'approche objet) :

```
   public String toString(){
      String res = "identification=" + this.getIdentification()+
            " description="+this.description;
      if (this instanceof StandAtelier)
         res += "\n Atelier - nb participants attendus="
             +((StandAtelier) this).getNbParticipantsAttendus();
      else if (this instanceof StandPermanent)
```

```
        res += "\n stand permanent";
      return res;    }
```

Expliquez en quoi le style n'est pas dans l'esprit de l'approche objet et proposez une écriture alternative en redistribuant le code sur les différentes classes. Reportez également la signature (entête) des méthodes dans le schéma UML.

Barême (4 pts)

- **1pt pour explication que : c'est non extensible (ce code doit être modifié si une nouvelle sous-classe est ajoutée), qu'il faut re-dispatcher dans les classes et ne pas faire des tests de type**
- **1pt pour méthode abstraite dans Stand**
- **1 pt pour méthode redéfinie dans StandAtelier**
- **1 pt pour méthode redéfinie dans StandPermanent**

```
// Dans Stand
public String toString(){
   String res = "identifcation=" + this.getIdentification()+
               " description="+this.description;
   return res;
}

// Dans StandAtelier
public String toString(){
    String  res =super.toString()+
       "\n Atelier - nb participants attendus="+this.getNbParticipantsAttendus();
   return res;
}

// Dans StandPermanent
public String toString(){
   String  res = super.toString()+"\n stand permanent";
   return res;
}
```

Question 4. Ecrivez en Java les méthodes nécessaires au calcul du prix de la location d'un stand (`prixLocation`) avec les informations suivantes (analysez les parties communes pour distribuer correctement le code entre les classes) :

- On suppose qu'un espace est associé au stand.
- Elles retournent un prix (elles ne l'affichent pas).
- Pour un stand atelier, le prix de location inclut :
  - le produit de
    - la durée de location en heures,
    - la surface associée,
    - le prix de base de la location par heure et par m^2.
  - le prix de la prestation en eau si celle-ci a été demandée,
  - les prix ci-dessus augmentent de 10% si l'espace est un espace intérieur.
  - 5 euros par participant au-dessus de 10 participants (c'est-à-dire 0 euros pour 10 participants, 5 euros pour 11 participants, 10 euros pour 12 participants, etc.)
- Pour un stand permanent, le prix de location inclut :
  - le produit de
    - la durée de location en heures,
    - la surface associée,

— le prix de base de la location par heure et par m^2.
— le prix de la prestation en eau si celle-ci a été demandée,
— les prix ci-dessus augmentent de 10% si l'espace est un espace intérieur.

Reportez également la signature (entête) des méthodes dans le schéma UML.

Barême (3 pts)

— **1pt pour réaliser la factorisation dans Stand et comprendre qu'il n'y aura qu'une redéfinition à faire**
— **1pt pour le calcul (même mal placé)**
— **1 pt pour méthode redéfinie dans StandAtelier**

```
// Dans Stand
   public double prixLocation(){
      double prix = this.dureeLocationEnHeures()
         * this.prixBaseLocationParHeureParM2
         * this.espaceAssocie.getSurfaceEnM2();
      if (this.prestationEauDemandee)
         prix += this.prixPrestationEau;
      if (this.espaceAssocie.isEstInterieur())
         prix = (1.1)*prix;
      return prix;
   }

// Dans StandAtelier
   public double prixLocation(){
      double prix = super.prixLocation();
         if (this.nbParticipantsAttendus > 10)
            prix += (nbParticipantsAttendus-10)*5;
         return prix;
   }
```

Question 5. Ecrivez en Java les méthodes nécessaires à la saisie des stands de tous les types (`void saisie (Scanner sc)`) en analysant les parties communes pour distribuer correctement le code entre les classes. Vous ne saisirez pas les attributs statiques dans cette méthode, ni le salon, ni l'espace associé au stand (pour simplifier). Reportez également la signature (entête) des méthodes dans le schéma UML.

Barême (2 pts)

— **1pt pour réaliser la factorisation dans Stand et comprendre qu'il n'y aura qu'une redéfinition à faire**
— **1 pt pour méthode redéfinie dans StandAtelier**

```
// Dans Stand
   public void saisie (Scanner sc){
      // pas les attributs statiques ni l'espace associé
      System.out.println("veuillez entrer l'identification du stand");
      identification = sc.next();
      System.out.println("veuillez entrer la description du stand");
      description = sc.next();
      System.out.println("veuillez entrer un booleen indiquant si le stand necessite de l'eau");
      prestationEauDemandee = sc.nextBoolean();
   }

// Dans StandAtelier
   public void saisie(Scanner sc){
      super.saisie(sc);
      System.out.println("Veuillez entrer la duree de l'atelier en heures");
```

```
      this.dureeAtelierEnHeures = sc.nextInt();
      System.out.println("Veuillez entrer le nombre de participants attendus");
      this.nbParticipantsAttendus = sc.nextInt();
   }
```

3 Représentation plus approfondie d'un salon

Question 6. Ecrivez en Java, dans la classe `Salon`, une méthode `ajoute` prenant en paramètre un stand. Si le stand se trouve déjà dans la liste ou s'il est déjà associé à un autre salon, un message d'erreur est affiché et l'ajout n'est pas effectué. Sinon, le stand est ajouté dans la liste des stands et le salon lui est associé. Reportez également la signature (entête) de la méthode dans le schéma UML.

Barême (2 pts)
— **1pt pour réaliser l'ajout et l'association du salon**
— **1pt pour les tests et le message d'erreur si besoin**

```
public void ajouteStand(Stand s){
   if (! listeStands.contains(s) && (s.getSalonAssocie()==this) ){
      listeStands.add(s);
   }
   else System.out.println("pb");
}
```

Question 7. Ecrivez en Java, dans la classe `Salon`, une méthode `prixTotalLocationsStands` retournant la somme des prix de location des stands. Reportez également la signature (entête) de la méthode dans le schéma UML.

Barême (3 pts)
— **0,5 pt pour la signature**
— **0,5 pt pour le retour de la valeur**
— **1pt pour réaliser correctement l'itération**
— **1pt pour le cumul des valeurs dans la somme**

```
public double prixTotalLocationsStands(){
   double pT = 0;
   for (Stand s : listeStands){
      pT += s.prixLocation();
   }
   return pT;
}
```

Question 8. Ecrivez en Java, dans la classe `Salon`, une méthode `grandStandsExterieurs` retournant la liste des stands dont l'espace associé se trouve en extérieur et possède une surface supérieure ou égale à 20 m^2. Reportez également la signature (entête) de la méthode dans le schéma UML.

Barême (3 pts)
— **0,5 pt pour la signature**
— **0,5 pt pour la création et le retour de la valeur de type ArrayList**
— **1pt pour réaliser correctement l'itération**
— **1pt pour le cumul des valeurs dans la ArrayList**

```
public ArrayList<Stand> grandStandExterieurs(){
   ArrayList<Stand> res = new ArrayList<Stand>();
      for (Stand s : listeStands){
         if (s.getEspaceAssocie()!=null
```

```
            && ! s.getEspaceAssocie().isEstInterieur()
            && s.getEspaceAssocie().getSurfaceEnM2()>=20)
         res.add(s);
         }
      return res;
   }
```

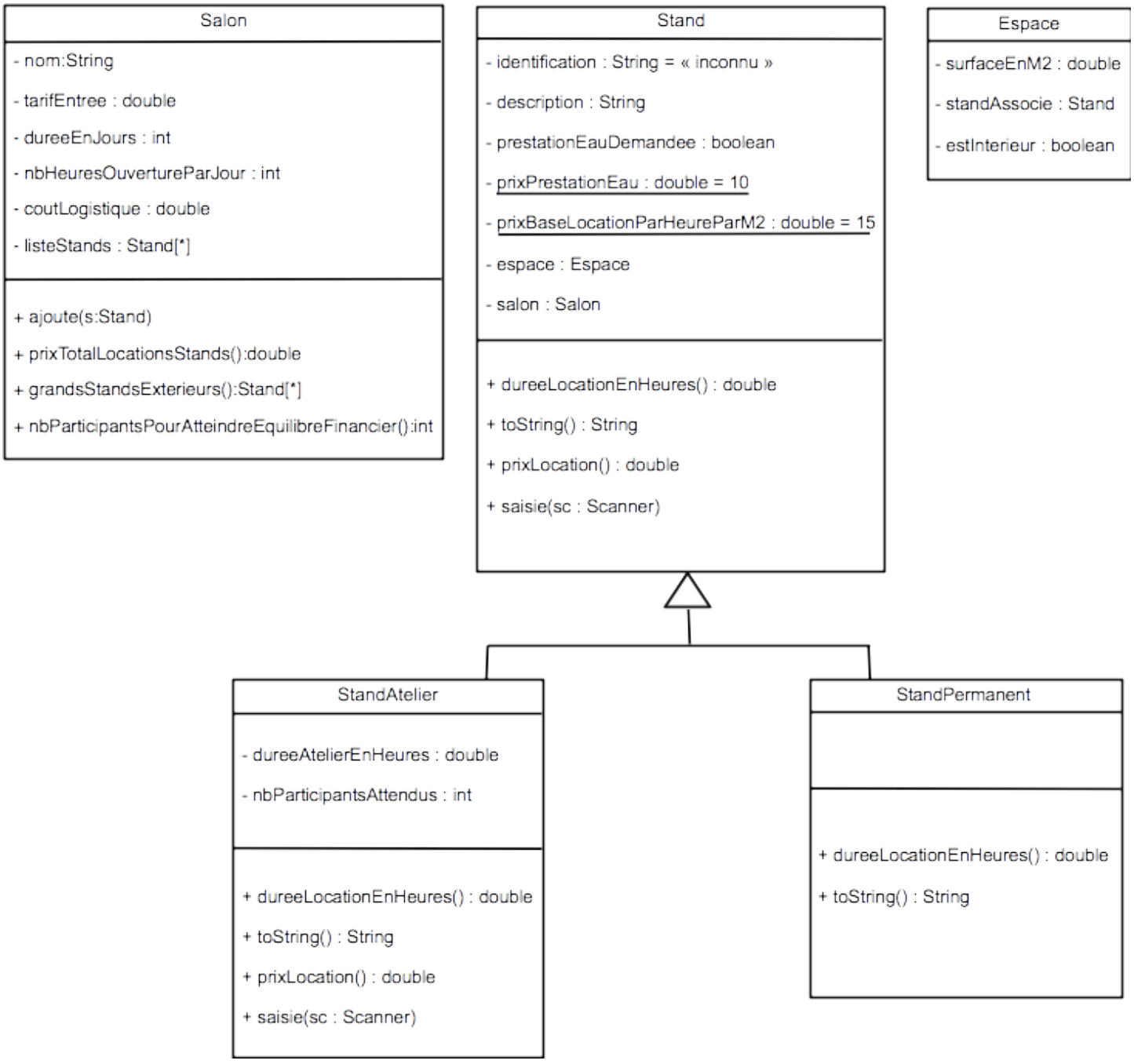
Question 9. Ecrivez en Java, dans la classe `Salon`, une méthode `nbParticipantsPourAtteindreEquilibreFinancier` retournant le nombre de participants nécessaires pour couvrir le prix de location des stands et le coût de la logistique. Reportez également la signature (entête) de la méthode dans le schéma UML.

Barême (2 pts)

```
public double nbParticipantsPourAtteindreEquilibreFinancier(){
   if (tarifEntree >0){
      return (this.coutLogistique+this.prixTotalLocationsStands())/tarifEntree;
   }
   else return -1;
}
```

4 Schéma UML à remplir

- Un exemple de syntaxe vous est donné pour les attributs et pour une méthode.
- '-' signifie "private", "+" signifie public).
- Les éléments représentés sont l'attribut `private String nom;` et la méthode `void ajoute(Stand s)`.
- `void` ne se représente pas en UML. Si la méthode possède un type de retour il est ajouté comme dans l'exemple Java `int methode(double d)` qui s'écrirait en UML `methode(d: double) : int`.
- Si un élément est static, il doit être souligné.
- Enfin si une classe est abstraite, son nom est surmonté du mot-clef abstract.


Salon
- nom:String
- tarifEntree : double
- dureeEnJours : int
- nbHeuresOuvertureParJour : int
- coutLogistique : double
- listeStands : Stand[*]
+ ajoute(s:Stand)
+ prixTotalLocationsStands():double
+ grandsStandsExterieurs():Stand[*]
+ nbParticipantsPourAtteindreEquilibreFinancier():int
Stand
- identification : String = « inconnu »
- description : String
- prestationEauDemandee : boolean
- prixPrestationEau : double = 10
- prixBaseLocationParHeureParM2 : double = 15
- espace : Espace
- salon : Salon
+ dureeLocationEnHeures() : double
+ toString() : String
+ prixLocation() : double
+ saisie(sc : Scanner)
Espace
- surfaceEnM2 : double
- standAssocie : Stand
- estInterieur : boolean
StandAtelier
- dureeAtelierEnHeures : double
- nbParticipantsAttendus : int
+ dureeLocationEnHeures() : double
+ toString() : String
+ prixLocation() : double
+ saisie(sc : Scanner)
StandPermanent
+ dureeLocationEnHeures() : double
+ toString() : String