Aucun document. Aucune machine. Barème indicatif. Parties indépendantes.
Une question peut toujours être traitée en utilisant les précédentes (traitées ou non).
Les morceaux de code Java devront être clairement présentés, indentés et commentés.

A la fin des années 1990, l'Université Paris 7–Denis Diderot décide de quitter le campus de Jussieu et de s'installer sur un nouveau campus composé de bâtiments à rénover et de bâtiments à construire. Des appels d'offre sont lancés et l'architecte Mad Bitree sort de ses cartons le projet suivant (retoqué pour des raisons de sécurité) : un bâtiment en forme d'arborescence, qui contient un unique point d'entrée (la *racine*) dont partent 2 nouveaux couloirs (*gauche* et *droite*). Au bout de chacun de ces couloirs partent éventuellement deux autres nouveaux couloirs (ou aucun) et ainsi de suite. Chaque couloir contient des salles (ou pas). Un schéma sommaire d'une partie du bâtiment :

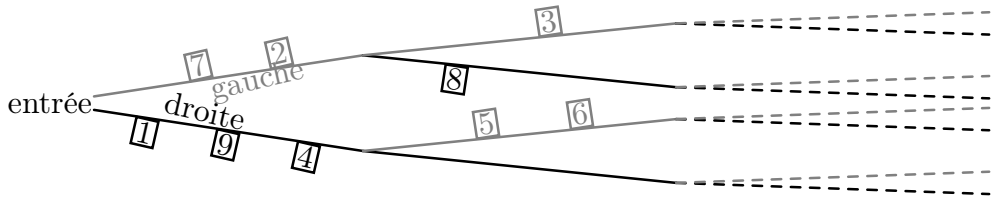

Figure 1 – Exemple de bâtiment.

Notre but est de modéliser un tel bâtiment. Pour cela, on introduit plusieurs classes, dont :
- `Salle` qui modélise une salle ;
- `Couloir` qui modélise un couloir ;
- `Batiment` qui modélise le bâtiment.

1 Une salle

Chaque salle possède certaines caractéristiques : un numéro d'identification unique (unicité sur tout le bâtiment), une capacité totale, un éventuel vidéoprojecteur, qui sont caractérisés par des attributs privés.

Par ailleurs son interface est la suivante :

```
public int getId();
public int getCapacite();
public void setCapacite(int cap);
public boolean possedeVideo();
```

Exercice 1. (1,5 points) Écrire le code de la classe `Salle` avec un constructeur permettant de spécifier la capacité et la présence éventuelle d'un vidéoprojecteur (et rien d'autre).

2 Les salles d'un couloir

Dans un couloir du bâtiment, il y a un nombre fini de salles. On veut représenter cet ensemble de salles sous forme d'une liste chaînée simple. L'ordre des salles dans une telle liste correspondra à l'ordre de parcours des salles dans le couloir. Par exemple pour le couloir à droite à partir de l'entrée, les salles doivent apparaître dans l'ordre 1–9–4.

Exercice 2. (1,5 points) Donner les attributs des types `CSalle` et `ListeSalles` correspondant respectivement à une cellule et à une liste. Faire le schéma correspondant à une liste de 3 salles.

Dans la suite de cette section, les méthodes demandées sont dans la classe `ListeSalles`.

Exercice 3. (0,5 points) Écrire une méthode qui teste si une liste est vide.

Exercice 4. (1 point) Écrire une méthode `public int numDansCouloir(int idSalle)` qui renvoie la position dans le couloir de la salle dont l'identifiant est donné en argument, si elle existe, une valeur prise par convention sinon. Par exemple, invoquée sur le couloir à droite de l'entrée, avec l'argument 9, elle doit renvoyer 2.

Exercice 5. (2,5 points)

(a) Écrire une méthode `public Salle salleAvecVideo()` qui renvoie une salle du couloir courant avec vidéoprojecteur (s'il y en a).

(b) Peut-on écrire dans la même classe une méthode de même nom `public Salle salleAvecVideo(int capMin)` ? (Justifiez votre réponse.) L'écrire (en changeant son nom si besoin) : elle doit renvoyer une salle du couloir courant avec vidéoprojecteur de capacité au moins celle donnée en argument (si elle existe).

Exercice 6. (3 points) Au bout de quelques années, on se rendra sûrement compte que le bâtiment ne contient pas assez de salles. Une solution consiste alors à couper en deux toutes les salles suffisamment grandes. Écrire une méthode `public int coupeSalles(int cap)` qui "coupe en deux" les salles du couloir courant dont la capactié est au moins celle fournie en argument. A partir d'une telle salle, on a deux nouvelles salles de capacité moitié qui possèdent toutes deux un vidéoprojecteur si et seulement si la salle de départ en contenait un (sinon aucune des deux n'en a). La première des deux salles dans le sens de parcours du couloir récupère l'identifiant de l'ancienne salle (celle qui a été coupée en deux), la deuxième récupère un nouvel identifiant. On a donc créé une salle supplémentaire. La méthode devra renvoyer le nombre de salles supplémentaires ainsi créées dans le couloir courant et modifier la liste de salles en conséquence.

3 Les couloirs et le bâtiment

Un couloir contient une liste de salles, des références vers les éventuels couloirs dont il est à l'origine et une référence vers le couloir dont il est issu. Le bâtiment contient une référence vers les 2 premiers couloirs.

Par ailleurs, on modifie la classe `Salle` en y ajoutant un attribut de type `Couloir` qui référence le couloir dans lequel se trouve la salle. On suppose également que cette salle possède la méthode `public Couloir getCouloir()`.

Exercice 7. (2 points) Donner les attributs des classes `Couloir` et `Batiment`.

Dessiner la structure de données correspondant à la figure 1, dans laquelle on se restreint aux 2 premiers niveaux (couloirs non pointillés).

Exercice 8. (2 points) Dans la classe `Couloir`, écrire une méthode `public int profondeur()` qui renvoie la distance d'un couloir à l'entrée du bâtiment (distance exprimée en nombre de couloirs à parcourir). Par exemple, les couloirs qui partent de l'entrée sont à distance 0 de l'entrée.

Dans la suite de cette section, les méthodes demandées sont dans la classe `Batiment`.

Exercice 9. (2 points) Écrire une méthode `grandeSalleAvecVideo(int capMin)` qui renvoie, si elle existe, la référence d'une salle de capacité au moins `capMin` possédant un vidéo-projecteur.

Exercice 10. (4 points) On veut écrire une méthode qui renvoie le cheminement le plus court pour aller de l'entrée à la salle d'identifiant donné en argument, sous la forme d'une chaîne de caractères composée d'une suite de `'g'` ("couloir gauche") et `'d'` ("couloir droit"), suivie de chiffres (représentant la position de la salle dans son couloir). Par exemple, dans le bâtiment représenté en figure 1, la salle 5 a pour cheminement `"dg1"`. On vous demande d'écrire cette méthode en suivant deux algorithmes différents :

1. Dans une première version `cheminementT2B(Salle salle)`, le parcours de l'arbre se fait de haut en bas, en partant de la racine pour trouver la salle.
2. Dans une deuxième version `cheminementB2T(Salle salle)`, le parcours de l'arbre de fait de bas en haut, en partant de la salle.

Attention : il s'agit de deux algorithmes différents, mais le résultat renvoyé doit être le même.