Nom :
Prénom :
Numéro d'étudiant :
Groupe :

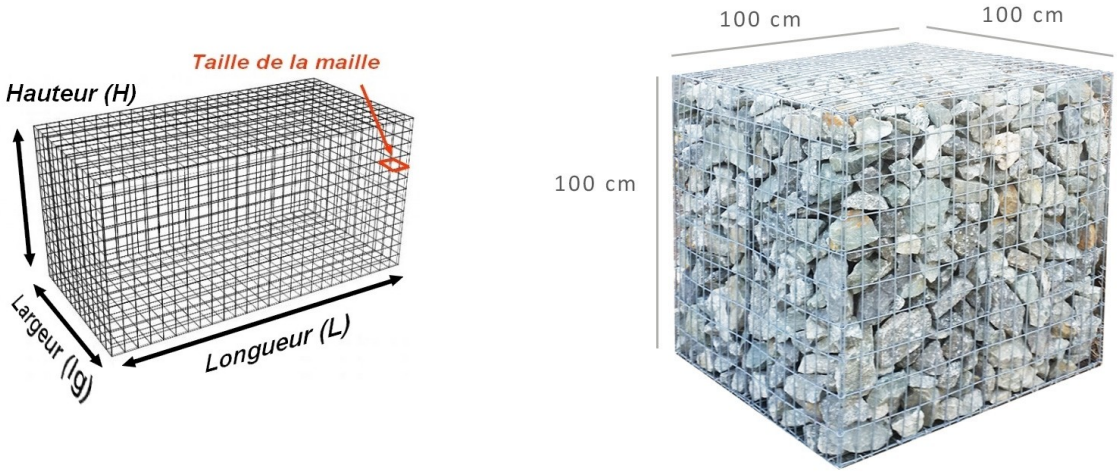
Contrôle continu

Tous documents sur support papier autorisés. Durée : 1h30.
L'ensemble des réponses sera à donner sur les feuilles d'énoncé. Ne pas dégrapher les feuilles.

On s'intéresse à un logiciel de simulation de murs de pierres à usage paysager. Toutes les dimensions de longueur sont exprimées en mètres, les volumes en m^3, les prix en euros.

1 Gabions

Dans cette section on s'intéresse aux gabions, qui permettent la contruction rapide de murs de pierres. Les gabions sont en effet des sortes de cages métalliques, que l'on peut remplir de pierres (voir figure 1). On se limite dans tout l'énoncé aux gabions et aux pierres de forme parallélipédique rectangle (ce sont donc des pavés droits).



(a) Gabion vide

(b) Gabion rempli de pierres

FIGURE 1 – Gabion vide (à gauche) et gabion cubique rempli (à doite)

On donne à la figure 2 un diagramme de classes partiel pour les gabions.

La classe `Dimensions` représente les dimensions d'un parallélépipède rectangle (pavé droit) ou de rectangles (dans ce cas la profondeur est nulle). La longueur est toujours la plus grande des trois dimensions (c'est-à-dire que longueur $\geq$ hauteur et longueur $\geq$ profondeur). La méthode `plusPetiteDimension` retourne la plus petite des 3 dimensions (c'est-à-dire soit la largeur soit la hauteur).

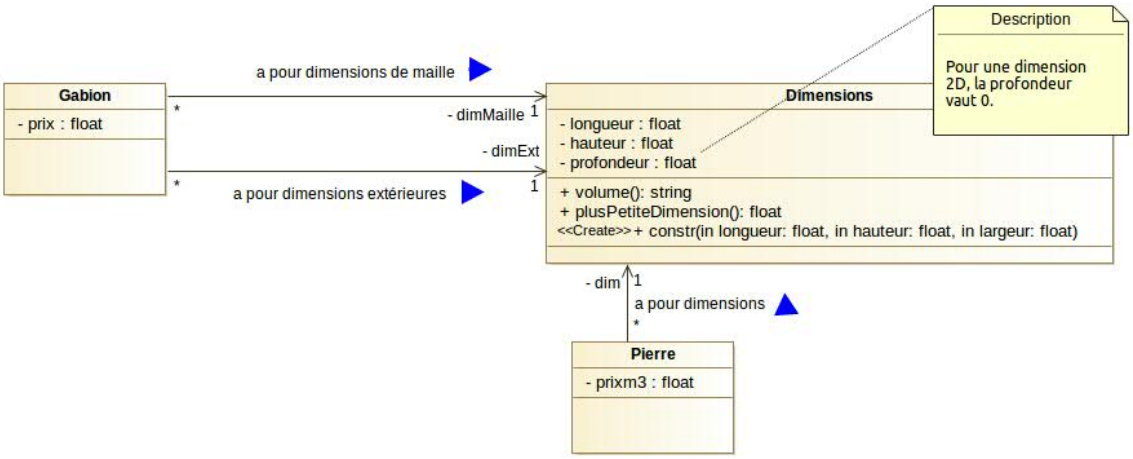


FIGURE 2 – Diagramme de classe partiel pour les gabions et les dimensions

On donne le code suivant.

```
public class Dimensions {
    private float longueur;
    private float hauteur;
    private float profondeur;

    public float getLongueur(){
        return longueur;
    }

    public Dimensions(float longueur, float hauteur, float profondeur) {
        this.longueur = longueur;
        this.hauteur = hauteur;
        this.profondeur = profondeur;
    }
}
```

Question 1. Donnez en Java pour la classe `Dimensions` : son entête, la déclaration de ses attributs, la déclaration et l'implémentation des méthodes `volume` et `plusPetiteDimension`.
Réponse à la question 1 :

```
    public float volume(){
        return longueur*hauteur*profondeur;
    }

    public float plusPetiteDimension(){
        if (hauteur>=profondeur){
            return profondeur;
        } else return hauteur;
    }
```

Un gabion a des dimensions extérieures (ce sont les dimensions de la cage métallique) et des dimensions pour les mailles : chaque face d'un gabion est une grille, les mailles sont les "trous" de la grille, elles sont rectangulaires ou carrées. Il est important de mettre dans un gabion des pierres qui

sont plus grandes que la taille de la maille, sinon les pierres vont sortir du gabion. On modélise ici des pierres avec la classe Pierre, qui définit juste le prix au m^3 de la pierre, et ses dimensions (on rappelle qu'on se limite ici aux pierres de forme parallélipédique rectangle (qui sont des pavés droits).

On suppose disposer du code Java suivant :

```
public class Gabion {
    private float prix;
    private Dimensions dimExt;
    private Dimensions dimMaille;

    public Gabion(float prix, Dimensions dimExt, Dimensions dimMaille) {
        this.prix = prix;
        this.dimExt = dimExt;
        this.dimMaille = dimMaille;
    }
    public float getPrix() {
        return prix;
    }
}
```

```
public class Pierre {
    private float prixm3;
    private Dimensions dimExt;

    public float getPrixm3() {
        return prixm3;
    }
    public Dimensions getDimExt() {
        return dimExt;
    }
    public Pierre(float prixm3, Dimensions dimExt) {
        this.prixm3 = prixm3;
        this.dimExt = dimExt;
    }
}
```

Question 2. Donnez en Java pour la classe `Gabion` l'entête et le corps d'une méthode `estCompatibleAvec` qui, pour une pierre donnée, dit si elle est compatible avec le gabion. On dira qu'une pierre p est compatible avec un gabion g si et seulement si la plus petite dimension de p est une fois et demie plus grande que la longueur de la maille de g.

Réponse à la question 2 :

```
public boolean estCompatibleAvec(Pierre p){
    return p.getDimExt().plusPetiteDimension()/p.getDimExt().getLongueur()>=1.5;
}
```

Question 3. Donnez en Java le code nécessaire pour déclarer une variable de type `Gabion`, et d'y affecter une nouvelle instance de gabion de dimensions extérieures $1m \times 1m \times 1m$, de dimensions de maille $0.01m \times 0.01m$ et de prix 30 euros.
Réponse à la question 3 :

```
Gabion g=new Gabion(30f,new Dimensions(1, 1, 1), new Dimensions(0.001f, 0.001f, 0));
```

2 Murs

On s'intéresse maintenant aux murs de pierres, une modélisation partielle est donnée à la figure 3. Un mur a des dimensions souhaitées. Un mur peut être ou pas un mur de clôture. On choisit pour un mur le type de pierre à utiliser. Le volume nécessaire de pierres est calculé par la méthode `calculVolumePierre`, on suppose disposer par la suite de l'implémentation de cette méthode. On distingue pour l'instant deux sous-classes de la classe (abstraite) `MurEnPierre` : `MursAGabion` et `MurMaçonné`. La spécialisation a été faite sur le critère du type de construction. Chaque mur à gabion est lié au type de gabion choisi, et la méthode `nbGabionsNecessaires` calcule et retourne le nombre de gabions nécessaires pour la réalisation du mur, on suppose disposer par la suite de l'implémentation de cette méthode. Chaque mur maçonné a une couleur de mortier.

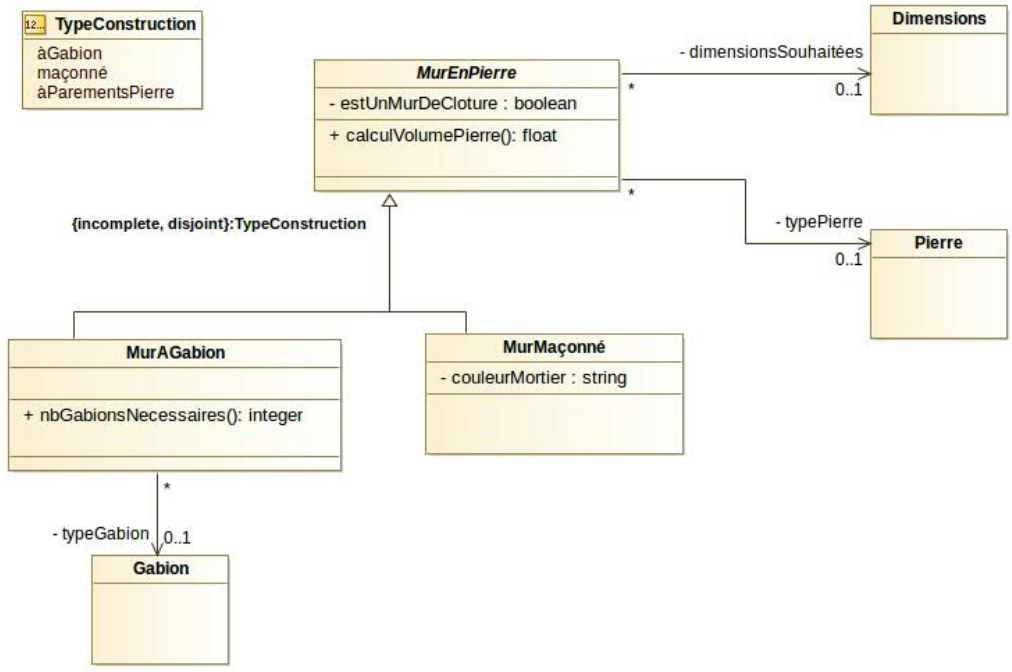


FIGURE 3 – Modélisation partielle des murs

Question 4. On souhaite disposer pour tous les murs d'une méthode `prixMo` qui retourne le prix de la main d'œuvre pour construire le mur. Dans le cas des murs à gabions, ce prix est de x euros par gabion nécessaire (où x est une valeur que l'on fixe ici arbitrairement à 200). Pour les murs maçonnés, le prix est de y euros par m^3 de pierres nécessaires (où y est une valeur que l'on fixe ici arbitrairement à 300). Écrire en Java, partout où elle est nécessaire dans la hiérarchie des murs, cette méthode `prixMo` (entête et corps). On précisera bien dans quelle classe est placé quel code.

Réponse à la question 4 :

```
// Dans MurEnPierre
public abstract float prixMo();
// Dans MurAGabion
public float prixMo() {
  return 200*nbGabionsNecessaires();
}
// Dans MurMaconne
public float prixMo() {
    return 300*calculVolumePierre();
}
```

Question 5. On souhaite disposer pour tous les murs d'une méthode `prixMateriel` qui retourne le prix du matériel nécessaire à la construction du mur. Dans le cas des murs à gabions, il s'agit du prix des pierres nécessaires ajouté au prix des gabions nécessaires. Dans le cas du mur maçonné il s'agit du prix des pierres nécessaires (on néglige le prix du mortier). Écrire en Java, partout où elle est nécessaire dans la hiérarchie des murs, cette méthode `prixMateriel` (entête et corps). On précisera bien dans quelle classe est placé quel code.
Réponse à la question 5 :

```
// Dans MurEnPierre
public float prixMateriel(){
    return calculVolumePierre()*typePierre.getPrixm3();
}
// Dans MurAGabion
public float prixMateriel(){
    return super.prixMateriel()+nbGabionsNecessaires()*typeGabion.getPrix();
    }
```

Question 6. On souhaite disposer pour tous les murs d'une méthode `prix` qui retourne le prix du mur. Pour les murs à gabions comme pour les murs maçonnés, ce prix est la somme du prix de la main

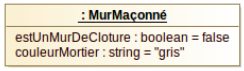
d'œuvre et du prix du matériel. Écrire en Java, partout où elle est nécessaire dans la hiérarchie des murs, cette méthode `prix` (entête et corps). On précisera bien dans quelle classe est placé quel code.
Réponse à la question 6 :

```
// Dans MurEnPierre
public float prix(){
    return prixMo()+prixMateriel();
    }
```

Question 7. Ecrire en Java des constructeurs paramétrés pour les classes `MurPierre` et `MurMaçonné`, ces constructeurs n'inialiseront pas la pierre choisie ni les dimensions souhaitées.
Réponse à la question 7 :

```
public MurEnPierre(boolean estDeCloture) {
    this.estDeCloture = estDeCloture;
    }
public MurMaconne(boolean estDeCloture, String couleurMortier) {
    super(estDeCloture);
    this.couleurMortier=couleurMortier;
}
```

Question 8. Donnez un diagramme d'instances représentant un mur maconné de mortier gris et qui n'est pas un mur de clôture. Ce mur n'aura pas encore de dimensions souhaitées ni de pierre choisie.
Réponse à la question 8 :



Type des attributs facultatif

3 Murs à parements de pierre

On souhaite ajouter à la modélisation existante celle des murs à parement de pierre. Ces murs sont des murs de pierre qui sont construits dans un matériau autre que la pierre, et qui sont juste recouverts de pierres ensuite (les pierres sont en quelque sorte collées sur le matériau avec du mortier). Ces murs sont décrits par le matériau central utilisé. De tels murs peuvent être à mortier apparent ou pas. Le matériau est décrit par son prix par m^3 et son type (brique ou parpaing), ainsi que par sa dimension unitaire (dimension d'une brique ou dimension d'un parpaing). Pour ces murs, on peut calculer sa profondeur approximative en ajoutant deux fois la profondeur de la pierre à la profondeur du matériau (on néglige la profondeur du mortier).

Question 9. Donnez en UML une modélisation des murs à parement de pierre (sans s'intéresser au calcul du prix de ces murs).
Réponse à la question 9 :