

TD n°3 – Composition

I. Segment

- 1) Dans votre projet Java TD3, créer un package segment et y créer une classe Segment dans laquelle un segment est **composé** de 2 points **distincts**, que l'on nommera *origine* et *extrémité*. Ces attributs sont des objets de la classe Point réalisée en TD1.
- 2) Inclure le projet java TD1 réalisé la semaine dernière dans le **Java Build Path** de votre projet d'aujourd'hui (TD3) afin de pouvoir instancier des objets de la classe point.Point du TD1.
- 3) Ajouter dans la classe la déclaration des attributs, la définition des constructeurs, des accesseurs en lecture, et une méthode qui affiche un Segment.
- 4) Détails en Java un programme de test de la classe Segment (une classe TestSegment). Compiler et exécuter avec succès.
- 5) Ajouter dans la classe Segment la méthode longueur qui retournera la longueur de l'objet courant.
- 6) Modifier TestSegment pour faire appel à la nouvelle méthode
- 7) Introduire dans la classe Segment deux nouvelles méthodes projX et projY permettant respectivement de calculer le projeté d'un segment support sur l'axe des abscisses et sur l'axe des ordonnées. (Mettre à jour TestSegment).

Attention à ne pas réécrire des méthodes disponibles dans les classes utilisées. Utilisez impérativement les méthodes de la classe Point quand cela s'y prête. Inversement, implémenter de nouvelles méthodes dans la classe Point si nécessaire pour Segment (ex : méthode estConfondu).

Garage

On souhaite développer une application permettant de gérer les stocks de voitures d'une usine d'assemblage.

On considère qu'une voiture est fabriquée à partir d'un moteur, de quatre roues et d'une roue de secours. Les voitures sont entreposées dans des garages. Un garage est décrit par un nom et une adresse et peut accepter un nombre de voitures quelconque défini lors de sa construction.

Le logiciel doit assurer la création et la maintenance de la liste des voitures entreposées dans les garages. Il doit permettre :

- d'enregistrer la fabrication d'une voiture en précisant les roues et le moteur qui ont permis son assemblage,
- de placer une voiture dans un garage,
- d'afficher la liste des voitures entreposées dans un garage,
- d'afficher la voiture la plus puissante entreposée dans un garage,
- d'afficher la valeur totale du stock des voitures entreposées dans un garage.

Dans votre projet TD3, créer un package garage dans lequel vous allez développer les classes décrites ci-dessous en suivant les indications ci-après.

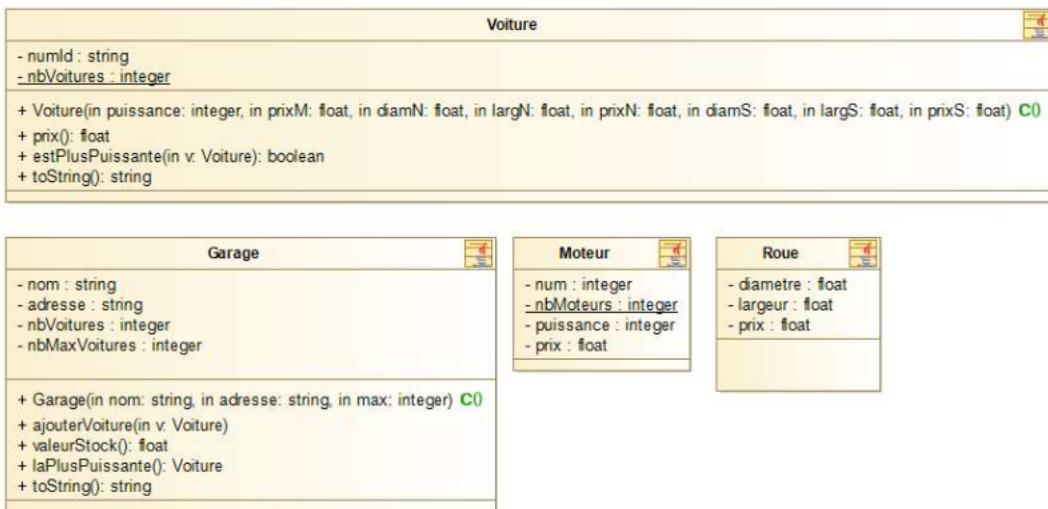


Figure 1: Diagramme de classes

- 1) Programmez les classes Roue et Moteur conformément aux spécifications de la figure (attributs et constructeurs) et aux indications suivantes :
 - Le constructeur de la classe Roue a 3 paramètres : diamètre, largeur et prix.
 - Définissez la méthode `toString` de la classe Roue afin qu'elle renvoie une chaîne de caractères de la forme : « diamètre x largeur ».
 - Le numéro d'un moteur (num) est un entier généré automatiquement lors de la création du moteur.
 - Le constructeur de la classe Moteur a 2 paramètres : puissance et prix.
 - Définissez la méthode `toString` de la classe Moteur afin qu'elle renvoie une chaîne de la forme «Moteur de puissance xx cv ».

- 2) Programmez la classe Voiture conformément aux spécifications de la figure (attributs et constructeurs) et aux indications suivantes :
 - Le numéro identificateur (numId) d'une voiture est une chaîne de caractères de la forme V1, V2 ... générée automatiquement lors de la création d'une voiture.
 - Définissez les variables d'instances représentant les liens entre la classe Voiture et les classes Moteur et Roue. Les roues normales seront représentées par un tableau de 4 éléments de type Roue.
 - La méthode `prix` renvoie le prix d'une voiture calculé ($= 4 * \text{prix roue normale} + \text{prix roue de secours} + \text{prix moteur}$).
 - Le constructeur d'une voiture admet 8 paramètres définis comme suit :
 - puissance et `prixM` : caractéristiques du moteur
 - `diamN`, `largN` et `prixN` : caractéristiques des quatre roues normales
 - `diamS`, `largS` et `prixS` : caractéristiques de la roue de secours.
 - Le moteur et les roues de la voiture doivent être instanciés dans le constructeur.
 - La méthode `estPlusPuissante` (Voiture `v`) renvoie la un résultat true si la voiture possède un moteur ayant une puissance supérieure à celle de la voiture `v` passée en paramètre.
 - Définissez la méthode `toString` afin qu'elle renvoie une chaîne semblable à celle de l'exemple suivant :

Voiture V0 Prix : 10510.0€
 Moteur de puissance 15 cv
 Roues normales : 17.0 x 165.0
 Roue de secours : 15.0 x 165.0

- 3) Définissez une classe TestGarage comportant une méthode main permettant de tester les méthodes de votre classe Voiture :
- Création d'une voiture voit1 ayant un moteur d'une puissance de 15 chevaux et un prix de 10000 euros, des roues normales d'un diamètre de 17 pouces, une largeur de 165 et un prix de 100 euros et une roue de secours d'un diamètre de 15 pouces, une largeur de 165 et un prix de 110 euros
 - Création d'une deuxième voiture voit2 ayant un moteur d'une puissance de 9 chevaux et un prix de 5000 euros des roues normales d'un diamètre de 17 pouces, une largeur de 155 et un prix de 200 euros et une roue de secours d'un diamètre de 15 pouces, une largeur de 155 et un prix de 220 euros
 - Affichage des caractéristiques de voit1 et voit2.
 - Affichage du prix de voit1.
 - Affichage de la voiture la plus puissante.
- 4) Programmez la classe Garage conformément aux spécifications de la figure (attributs et constructeurs) et aux indications suivantes :
- La liste des voitures stockées dans le garage sera représentée par un tableau de voitures.
 - La méthode ajouterVoiture(Voiture v) permet d'ajouter au garage une voiture v passée en paramètre. Si le nombre de voitures garées dans le garage est déjà égal à la capacité du garage (nbMaxVoitures), l'ajout doit s'avérer impossible et un message d'erreur doit être affiché.
 - La méthode valeurStock() renvoie la somme des prix des voitures garées dans le garage.
 - La méthode laPlusPuissanteVoiture() renvoie la voiture ayant le moteur le plus puissant.
 - La méthode toString() retourne une chaîne de caractères contenant les caractéristiques d'un garage sous un format semblable à l'exemple suivant :

```

GaragePARADISO(Nice) Capacité : 50
*****
Voiture V0 Prix : 10510.0€
Moteur de puissance 15 cv
Roues normales : 17.0 x 165.0
Roue de secours : 15.0 x 165.0

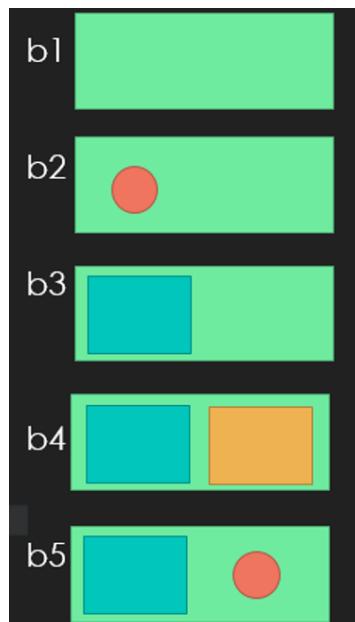
Voiture V1 Prix : 6020.0€
Moteur de puissance 9 cv
Roues normales : 17.0 x 155.0
Roue de secours : 15.0 x 155.0

La voiture la plus puissante est la voiture V0
La valeur totale du stock de voitures est de 16530.0€
  
```

- 5) Complétez la méthode main de votre classe TestGarage afin qu'elle effectue les actions suivantes :
- Création d'un garage ayant pour nom "PARADISO" et pour adresse "Nice".
 - Placement des voitures voit1 et voit2 dans le garage Paradiso.
 - Affichage de la liste des voitures du garage, du numéro identificateur de la voiture la plus puissante du garage ainsi que du total de la valeur du stock de voitures.

Boite

Nous souhaitons définir une classe représentant une boite, pouvant contenir un objet ainsi que d'autres boites. Un exemple de l'utilisation des boites est présenté dans la figure ci-dessous.



La classe Objet est donnée dans Moodle. Le but est d'implémenter la classe Boite, elle comprend :

- **Constante** : Créer une constante qui correspond au nombre maximum de boites pouvant être contenues dans une boite. Ce nombre est fixé à 5 et ne peut pas changer de valeur.
- **Attributs** : Les instances de Boite sont définies par quatre attributs : sa couleur (du type *java.awt.Color*), son contenu composé d'un objet et d'un tableau de boites, et le nombre de boites contenues dans le tableau.
- **Constructeurs** : Il y aura 4 constructeurs :
 - a. un prenant seulement une couleur en paramètre, signifiant que la boite est vide.
 - b. un prenant une couleur et un objet (le tableau de boites restant vide),
 - c. un prenant une couleur et une boite : le tableau de boites ne contiendra que cette boite, et le nombre de boites est ajusté,
 - d. un prenant une couleur, un objet, et une boite. Ce constructeur initialise alors une boite contenant un objet et une autre boite.
- **Méthodes** :
 - e. `getObjet()` et `getCouleur()` Ce sont des accesseurs, ils renvoient la valeur de l'attribut correspondant.

- f. `contientObjet(Objet o)` Cette fonction vérifie si la boite courante contient l'objet passé en paramètre ou non.
 - g. `estVide()` Cette fonction vérifie si la boite courante est vide ou non. La boite est vide si elle ne contient ni objet ni boîte, par contre elle peut avoir une couleur.
 - h. `ajouteBoite(Boite b)` Cette fonction modifie l'état de la boite courante en ajoutant la boîte b à son contenu s'il reste de la place. Sinon, il ne se passe rien.
- 1) Ecrire le code source Java de la classe Boîte.
 - 2) Donner le code de la méthode main permettant de reproduire les 5 boîtes de la figure de la page 3.

Les couleurs à utiliser sont Color.green, Color.blue, Color.red, et Color.yellow.

Nous considérons maintenant la classe `Objet` en Annexe de ce sujet. Nous souhaitons gérer une exception dans le cas où la couleur spécifiée dans la méthode `changeCouleur` est identique à celle de l'objet courant, au lieu de simplement afficher un message d'erreur.

- 3) Modifier la méthode `changeCouleur(Color c)` de `Objet` et lever une exception dans le cas où la couleur en paramètre est identique à celle de l'objet courant. Donner le code source complet (signature + contenu) de la nouvelle version de cette méthode.

Fleuriste

Vous devez mettre au point un programme pour la gestion du stock d'un magasin de fleurs. Le fleuriste vend des fleurs en bouquets. Les fleurs sont livrées au magasin à l'unité dans l'arrière-boutique et sont arrangées en bouquets. Un bouquet est un assemblage d'un certain nombre de tulipes, œillets et roses (exemple : 10 tulipes et 5 roses). Chaque type de fleur est caractérisé par un prix à l'unité (en euros). Le prix d'un bouquet est la somme des prix des fleurs qui le composent majoré de 15% (majoration due au travail de conception du bouquet).

Le fichier `TestBouquet` contenant une fonction main de mise en œuvre des classes vous est fournie dans *Moodle*.

- Proposez un modèle objet pour représenter un bouquet qui soit cohérent avec le main donné dans l'énoncé. Vous devez ainsi inclure une méthode `prix()` renvoyant le prix du bouquet.
- Proposez une classe pour représenter le stock (l'arrière-boutique) du magasin de fleurs. Donnez les attributs et écrivez la méthode `valeurStock()` qui renvoie la valeur du stock courant du magasin.
- Ecrivez la méthode `bouquetFaisable(Bouquet b)` de la classe Stock qui renvoie vrai si l'état du stock permet la confection du bouquet b donné en paramètre et faux sinon.