TD n°3 – Composition, Exceptions

Segment

1) Créer un nouveau package 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 (TD2) 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 des méthodes `equals, clone, toString`.
4) Détailler 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. (Modifier `TestSegment` pour faire appel à la nouvelle méthode)
6) 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.

Gestion des exceptions :

1) Dans cette classe, **lever une exception** dans le cas où les 2 Points donnés pour construire le Segment sont confondus.
2) Quelles autres méthodes de la classe Segment peuvent alors déclencher une exception ? Ajouter les déclarations adéquates afin de **propager l'exception** jusqu'au *main* de la classe `TestSegment`.
3) Mettre à jour `TestSegment` afin de **traiter les exceptions**.

Fleuriste

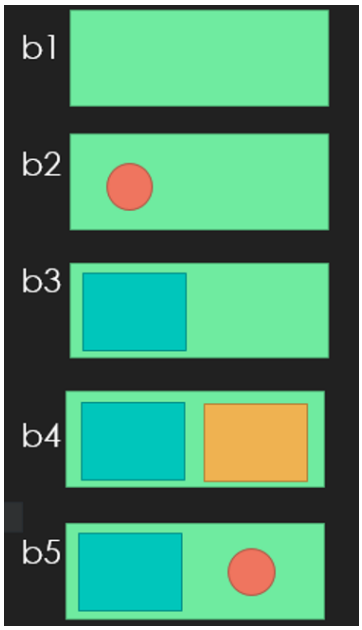
Vous devez mettre au point un programme pour la gestion du stock d'un magasin de fleurs. Le fleuriste vend des fleurs en bouquets. Les fleurs sont livrées au magasin à l'unité dans l'arrière-boutique et sont arrangées en bouquets. Un bouquet est un assemblage d'un certain nombre de tulipes, œillets et 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 *SupportCours*.

- 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.

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 en Annexe. 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 boite, par contre elle peut avoir une couleur.
    h. `ajouteBoite(Boite b)` Cette fonction modifie l'état de la boite courante en ajoutant la boite 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 `Boite` en vous aidant du squelette ci-dessous et de la classe `Objet` donnée en Annexe.

2) Donner le code de la méthode main permettant de reproduire les 5 boites 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.