Projet Programmation Orientée Objet 2014 - 2015

Description du sujet

L'objectif de ce projet est de programmer une application de gestion de découpe de panneaux de bois. Vous sous-traitez pour une entreprise (fictive) qui effectue des découpes de rectangles de bois spécifiées par des clients sur des planches qu'elle achète chez des fournisseurs. Le problème de cette entreprise est qu'elle ne dispose d'aucune méthode automatique pour effectuer ces découpes, et perd donc du bois. Vous devez leur livrer une application qui va permettre d'automatiser le calcul des découpes et permettre ainsi une économie substantielle sur les achats de planches.

Spécification des fournisseurs

Les planches de bois peuvent provenir de plusieurs fournisseurs et ont donc une longueur et largeur qui peuvent varier. Elles possèdent en outre un prix unitaire. Les différents types de planche sont définis dans un fichier XML qui revêt la forme suivante :

```
<fournisseurs>
    <fournisseur id="1" largeur="140" longueur="320" prix="3,25"/>
    <fournisseur id="2" largeur="320" longueur="320" prix="8"/>
</fournisseurs>
```

Chaque spécification est donnée selon la façon suivante :

- `id` : identifiant numérique d'un fournisseur
- `largeur` : largeur de la planche à découper en cm
- `longueur` : longueur de la planche à découper en cm
- `prix` : prix unitaire de la planche en euros

La spécification des fournisseurs est telle que la largeur est inférieure ou égale à la longueur.

Spécification des commandes clients

Les commandes des clients sont spécifiées dans un fichier XML qui revêt la forme suivante :

```
<commandes>
    <commande id="1" largeur="12" longueur="14" quantite="5"/>
    <commande id="2" largeur="6" longueur="48" quantite="10"/>
    <commande id="3" largeur="9" longueur="31" quantite="15"/>
    <commande id="4" largeur="150" longueur="160" quantite="1"/>
</commandes>
```

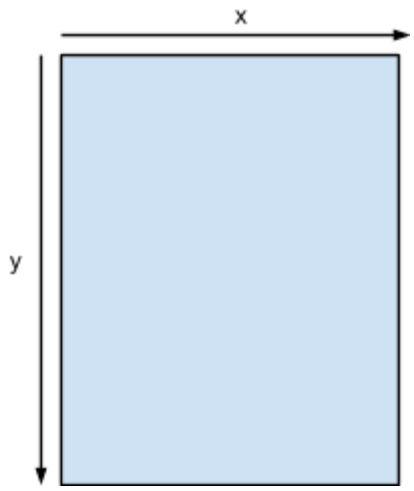
Un élément `commandes` englobe les commandes d'un unique client. Cet ensemble de commandes devra être traité en même temps. Chaque commande est spécifiée de la façon suivante à l'aide d'un élément `commande` qui contient les attributs suivants :

- `id` : identifiant numérique et unique de la commande.
- `largeur` : largeur de la planche en cm
- `longueur` : longueur de la planche en cm
- `quantite` : nombre de planche de largeur et longueur données à fournir

La spécification des commandes est telle que la largeur est inférieure ou égale à la longueur. En outre il ne peut y avoir deux commandes avec un identifiant différent et une largeur et longueur égales.

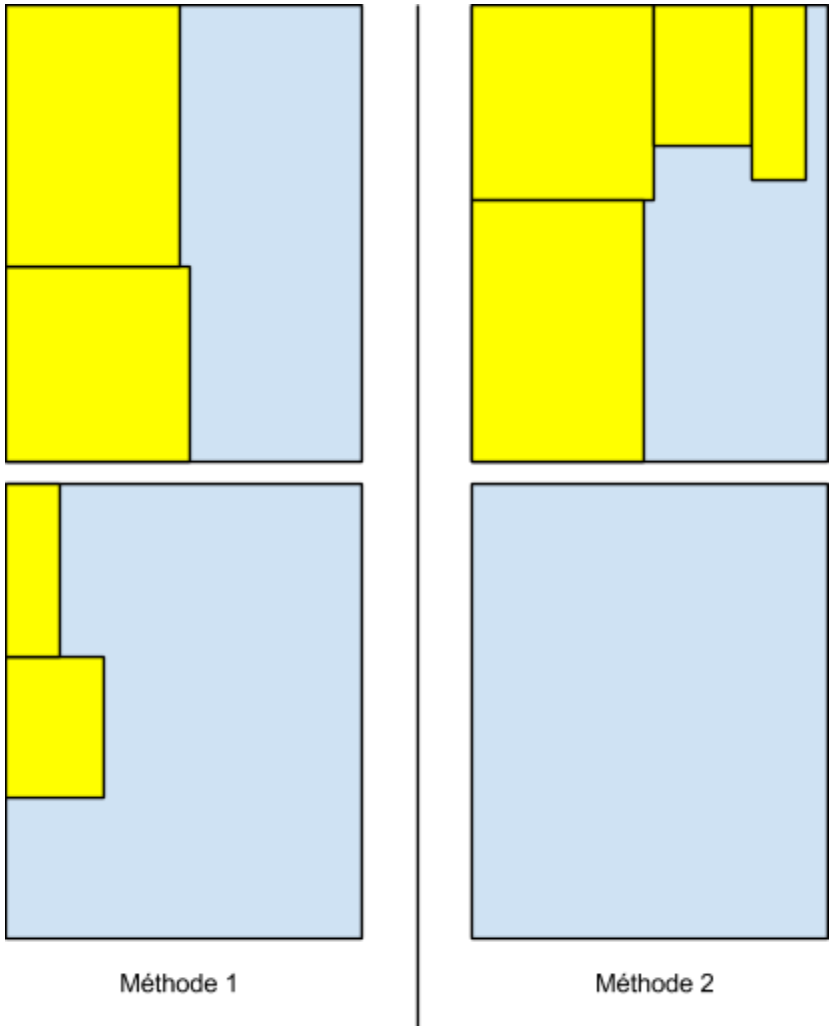
Planification des découpes

Vous devez implémenter deux méthodes différentes de planification des découpes. Pensez à éviter au maximum toute duplication de code. Vous devez favoriser la réutilisabilité et l'interchangeabilité des algorithmes (pour changer l'algorithme de découpe, les changements à appliquer doivent être minimaux). Les pièces qui seront découpées sur la *planche* sont nommées par la suite *découpe.* La *planche* est l'élément amené par le fournisseur. La figure suivante montre un exemple de planche. La largeur de la *planche* correspond à la coordonnée *x* et la longueur à la coordonnée *y*. Pour une découpe `d` donnée, la largeur est notée `l(d)` et la longueur `L(d)`. La longueur de la planche est notée `L(P)` et la largeur de la planche `l(P)`.



Méthode 1

La méthode 1 est simple mais peu efficace. Tout d'abord les découpes sont ordonnées par ordre de longueur décroissante (en cas d'égalité par largeur décroissante). La première découpe `d0` est placée, si possible, à la coordonnée `(0,0)`. Ensuite `d1` est placée à `(0,L(d0))` et ainsi de suite. S'il n'est pas possible de placer une découpe sur la planche en cours car la longueur restante est trop petite, il faut alors passer sur une nouvelle planche.



Méthode 2

La méthode 2 est théoriquement plus efficace. Le principe est de classer d'abord les découpes par largeur décroissante (en cas d'égalité par longueur décroissante). La première découpe `d0` est placée, si possible, à la coordonnée `(0,0)`. Ensuite les découpes suivantes sont parcourues par ordre de largeur décroissante et la première découpe `di` vérifiant `L(di) < L(d0)` et `l(di) < l(P) - l(d0)`. Cette découpe est alors placée à la coordonnée `(l(d0),0)`. Ensuite les découpes suivantes sont parcourues par ordre de largeur décroissante pour trouver une découpe `dj` telle que `L(dj) < L(d0)` et `l(dj) < l(P) - l(d0) - l(di)`. Elle est placée à la coordonnée `(l(d0)+l(di),0)`. Cette opération est répétée jusqu'a ce que l'on ne trouve plus de découpe de longueur inférieure à `d0` et rentrant sur la largeur de planche restante. Quand c'est le cas, on passe à la découpe la plus large restante `dk` qui est placée à la coordonnée `(0,L(d0))` est le processus reprend. Si il n'est pas possible de placer cette découpe sur la planche en cours car la longueur restante est trop petite (si `L(dk) > L(P) - L(d0)`), il faut alors passer sur une nouvelle planche.

Simulation

Votre programme doit permettre de simuler la commande d'un client étant donné une liste de fournisseurs. Votre programme doit permettre de gérer efficacement un très grand nombre de fournisseurs. Il doit produire en sortie pour chaque fournisseur :

1. Un fichier XML qui décrit les *découpes* avec la méthode 1
2. Un fichier XML qui décrit les *découpes* avec la méthode 2
3. Un fichier SVG qui montre graphiquement les *découpes* obtenues par la méthode 1
4. Un fichier SVG qui montre graphiquement les *découpes* obtenues par la méthode 2

La ligne de commande suivante doit effectuer la simulation d'une commande d'un client (spécifiée dans le fichier `commandes.xml`) compte tenu d'une liste de fournisseurs (spécifiée dans le fichier `fournisseurs.xml`).

```
java gestion.Devis commandes.xml fournisseurs.xml
```

Votre programme génère donc un total de `4 * n` fichiers, `n` étant le nombre de fournisseurs. Voici la convention de nommage de ces fichiers (dans l'ordre, pour le fournisseur 1) :

1. results.f1.m1.xml
2. results.f1.m2.xml
3. results.f1.m1.svg
4. results.f1.m2.svg

Voici un extrait d'un fichier résultat :

```
<simulation fournisseur="1" prix="3.25" rejets="1" algorithme="1">
  <rejets>
    <rejet commande="4"/>
  </rejets>
  <planches>
      <planche id="1">
            <decoupe commande="1" x="0" y="0"/>
            <decoupe commande="2" x="0" y="14"/>
            <decoupe commande="3" x="0" y="62"/>
      </planche>
      [...]
  </planches>
</simulation>
```

Vous pouvez créer les fichiers SVG en même temps que les fichiers de résultats, ou choisir

de les produire à partir des fichiers de résultats.

Un élément `simulation` englobe le document. Les attributs de cet élément sont :

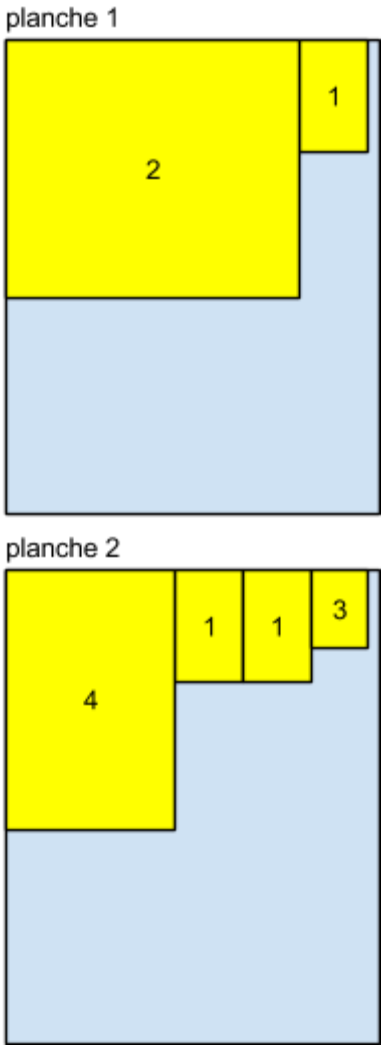
- `fournisseur` : l'identifiant du fournisseur pour qui la simulation a été faite.
- `prix` : le prix total du montant de la commande chez ce fournisseur. Il correspond au prix unitaire d'une planche multiplié par le nombre de planches utilisées.
- `rejet` : le nombre de commande non placées
- `algorithme` : le numéro de l'algorithme utilisé

L'élément `simulation` contient ensuite un élément `rejets` et un élément `planches` qui contiennent respectivement plusieurs éléments `rejet` et `planche`. Les éléments `planche` ont un attribut `id` numérique et unique correspondant aux différentes planches utilisées pour placer les commandes. Cet élément contient autant d'éléments `decoupe` qu'il y a de rectangles à découper sur la `planche`. Les attributs de cet élément `decoupe` sont les suivants :

- `commande` : identifiant de la commande à laquelle appartient cette découpe
- `x` : la coordonnée x de la découpe sur la planche (voir figure suivante pour le repère)
- `y` : la coordonnée y de la découpe sur la planche (voir figure suivante pour le repère)

L'élément `rejet` signifie que la commande dont l'identifiant est spécifiée à l'aide de l'attribut `commande` n'a pas été traitée car les dimensions souhaitées ne peuvent être satisfaites par les produits du fournisseur.

Le fichier SVG général doit ressembler à ceci (attention cet exemple ne correspond pas aux fichiers XML donnés en exemple dans le sujet) :



API

Pour parser les fichiers XML vous devez utiliser l'API `XMLStreamReader`.
Pour générer le SVG vous devez utiliser l'API `XMLStreamWriter`.
Une API (*Application Programming Interface*) est une interface fourni pour un programme extérieur, généralement de librairies. Pour chacune de ces API, les services (méthodes) qu'il est possible d'utiliser depuis l'extérieur sont exposés (via la javadoc notamment). Encore une fois, un exemple d'encapsulation des données.
Ces API sont répandues et vous devrez essayer de trouver vous-même comment utiliser cette API. Cela vous fera prendre en main une API externe.
Vous pourrez trouver des informations sur le format SVG ici. Ce format peux être lu par la majorité des navigateurs web.
Pensez en général à éviter toute redondance de code, réfléchissez à la conception avant de coder. Pensez aux évolutions de votre système. Est-ce qu'intégrer un nouvel algorithme peut se faire "facilement" au sein de votre programme ?

Bonus

Si vous avez répondu à toutes les exigence du projet, vous pouvez réfléchir à un nouvelle méthode de découpe et l'implémenter. Cette nouvelle méthode de découpe doit être plus efficace que les deux existantes.
Vous devez fournir des exemples montrant l'efficacité de votre méthode par rapport aux deux existantes. Notez que l'architecture de votre application doit vous permettre d'intégrer cette nouvelle méthode de découpe avec très peu de modification.

Contraintes

Votre projet doit respecter les contraintes énoncées ci-dessous. Une pénalité de 2 points sera appliquée aux projets ne les respectant pas.

Architecture

Le paquetage par défaut de votre application est `gestion`.
Les sources doivent se trouver dans un répertoire nommé `src`, les tests dans un répertoire `test`, les bibliothèques dans un répertoire `lib` et les binaires dans un dossier `bin`.
Votre application doit se lancer avec la commande suivante à partir du répertoire `bin` :

```
java gestion.Devis commandes.xml fournisseurs.xml
```

Code

- Votre projet contient plusieurs paquetages.
- Les noms des paquetages sont intégralement en minuscules.
- Les noms des classes et interfaces commencent par une majuscule.
- Les noms des méthodes, des attributs et des variables commencent par une minuscule.
- Utilisez le *camel case* pour les noms des classes, interfaces, méthodes et attributs.
- Les attributs ont une visibilité `private` ou `protected`. Des accesseurs permettent si nécessaire d'accéder en lecture et/ou écriture aux attributs.
- La visibilité des méthodes dépend de leur utilisation.
- Les constructeurs sont définis uniquement s'ils sont nécessaires.
- Le code de votre projet doit être commenté. Explicitez les méthodes non triviales.

Évaluation

Ce projet est à réaliser par groupe de 3 étudiants. Vous serez évalués sur :

- L'architecture de l'application
- La modularité et l'extensibilité de l'application.
- L'utilisation des concepts fondamentaux de la programmation orientée objet.
- La robustesse de votre application.

Livrables

Il sera demandé aux étudiants de fournir une archive contenant le code source du programme. Le schéma de nommage de l'archive doit être le suivant (pénalité de 1 point en cas de non respect) : **pg220-2014-LOGINS.zip**, où LOGINS est la liste des logins des membres du groupe, classée par ordre alphabétique et séparés par le symbole _.
L'archive est à déposer, avant le **5 Janvier 2015 à 23H**, via la plateforme http://moodle.ipb.fr.