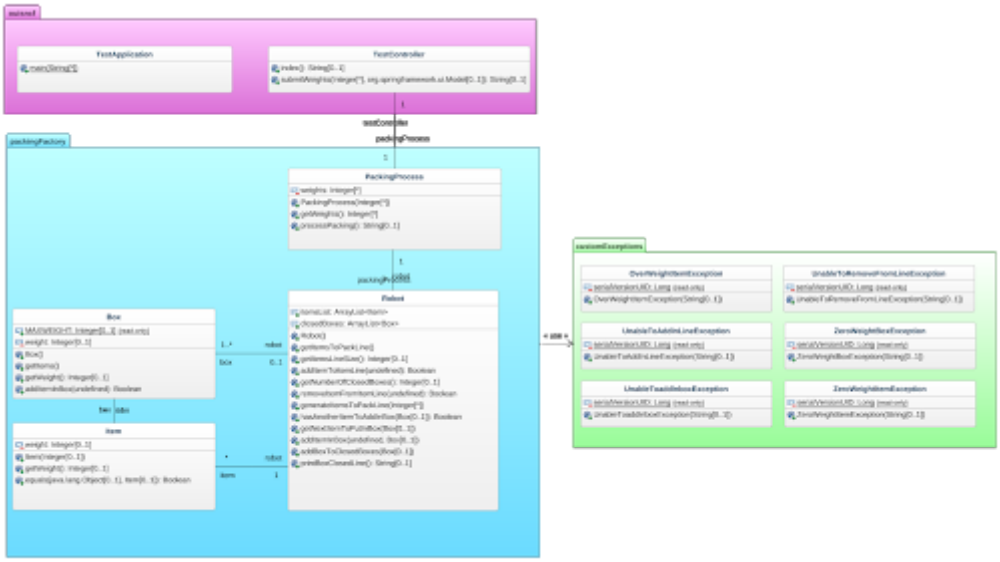
Application d'empaquetage pour le test technique ouisncf

@auteur: Lucie Vigreux

Architecture de l'application

Diagramme de classes



![Diagrammes de classes]

Diagramme généré avec genmymodel: https://www.genmymodel.com/fr

Une version du diagramme en png est présente à la racine du projet.

Principe de l'algorithme

L'algorithme que j'utilise diffère légèrement de celui donné en exemple.

Version de l'algorithme de l' exemple: Dans l'exemple fourni, pour la version non optimisée, le robot met l'objet suivant dans le carton si le poids maximum de 10 n'est pas dépassé. Dans sa version optimisée, il fait de même, mais si le poids du carton ne

dépasse pas 10 il va prendre l'item suivant lui permettant de maximiser le poids dans le carton. Ce qui permet de passer de 10 à 8 cartons utilisés.

Algorithme choisi: L'algorithme que j'ai choisi de mettre en oeuvre met le premier article dans le carton, puis va directement chercher l'article le plus lourd possible pour maximiser le poids du carton, et ce jusqu'à ce que la taille maximale du carton soit atteinte. Dans ma version optimisée on utilise également de 8 cartons, la différence est que je suis sure que pour chaque carton la taille est maximisée indépendamment de l'ordre des items présents à empaqueter. La répartition des items dans les cartons est donc légèrement différente que celle montrée dans l'exemple. Quand il n'y a plus d'objets à mettre dans le carton on le ferme.

L'application

Technologies utilisées:

L'application a été crée en utilisant spring, maven, java et docker Les templates HTML utilisent thymeLeaf et un tout petit peu de bootstrap. Pour les tests j'utilise Junit et le plugin Jacoco .

Utiliser l'application

Pour lancer l'application

Vous avez deux choix:

Avec Docker:
Tout peut se faire par maven puisque le plugin docker pour maven est installé dans le pom.xml

Générer l'image Docker:

```
mvn install dockerfile:build
```

Lancer un conteneur avec l'image générée:

```
docker run -p 8080:8080 -t springio/ouisncf-lucie-test
```

L'application est disponible à l'adresse: http://localhost:8080

Sans Docker:

Installer les dépendances:

```
mvn install
```

Lancer avec springboot:

```
mvn spring-boot:run
```

L'application est disponible à l'adresse: http://localhost:8080

Générer la javadoc:

La doc est taggée pour pourvoir être générée avec la commande:

```
mvn javadoc:javadoc
```

La javadoc est disponible dans le dossier:

```
/racine-de-lappli/target/site/apidocs
```

Cliquer sur `index.html` par exemple pour naviguer dans la doc.

Lancer les tests et générer le rapport:

```
mvn test
```

La couverture totale de tests est de 80%.

Le rapport complet est disponible dans le dossier:

```
/racine-de-lappli/target/site/jacoco
```

Cliquer sur `index.html` par exemple pour naviguer dans le rapport généré.

Lancer un processus d'empaquetage:

Dans le formulaire écrire sous la forme 1,2,3,4 la liste des poids des items souhaités. La répartition est donnée après avoir soumis la liste.

Pour les poids en exemple:
[1, 6, 3, 8, 4, 1, 6, 8, 9, 5, 2, 5, 7, 7, 3]

Répartition des objets dans les cartons:
91/82/81/73/73/64/6/55/

8 cartons utilisés également.

On voit la différence de répartition, pour la première boite par exemple, le premier objet ayant un poids de 1 le robot a tout de suite cherché à maximiser le poids dans la boite en mettant un objet de poids 9, plutôt que d'abord mettre les objets voisins et ensuite,si possible chercher à optimiser le poids ensuite.
Le poids est ici directement optimisé.

> Written with [StackEdit](https://stackedit.io/).