Approche Orientée-Objet

Cahier des charges

1 Introduction

L'objectif du travail demandé est de développer un éditeur de formes géométriques (Figure 1) pouvant être soit des formes simples (rectangle ou oval, par exemple) soit des compositions d'autres formes (union, intersection ou différence, par exemple). En particulier, en prenant un domaine [0..xmax]*[0..ymax], une forme pourra être dessinée en représentant uniquement les points (x,y) qui appartiennent à cette forme (ex. les points vérifiant la propriété $(x_1 \leq x \leq x_2) \wedge (y_1 \leq y \leq y_2)$ pour un rectangle définit par deux extrémités (x_1, y_1) et (x_2, y_2)).

Figure 1: Editeur de formes

Pour procéder, commencer par définir les différents types de formes avec une opération permettant de tester si un point (x, y) appartient à une forme. Ajouter aussi deux opérations pour: 1) déplacer une forme de (dx, dy), et 2) redimmensionner une forme de (kx, ky), et une opération supplémentaire pour avoir la description d'une forme (utiliser une description "arborescente" comme illustrée ci-dessous). Enfin, définir aussi un programme permettant d'afficher une forme sur la sortie standard: un point appartenant à une figure pouvant se représenter simplement par un `X` ; utiliser ce programme pour tester que tous les éléments précédents fonctionnent correctement.

```
+ union
  + rectangle(0,0,5,2)
  + difference
    + rectangle(2,0,6,2)
    + rectangle(3,1,5,3)
```

Dans un deuxième temps, ajouter la possibilité de pouvoir sauvegarder une forme dans un fichier ou de la restaurer (voir "persistence"). Permettre ensuite une sauvegarde sur une machine distante (voir "rmi") en définissant les opérations `Server.save(figure)` et `model = Server.load()`.

Dans un troisième temps, définir une interface graphique comme par exemple celle illustrée sur la Figure 1. L'interface devra être simple/conviviale et montrer toutes les fonctionnalités proposées par le logiciel. En particulier, on demande d'ajouter une fonctionalité "originale" n'ayant pas été demandée dans le cahier des charges (ex. ajout de couleurs, possibilité d'éditer les propriétés d'une forme, etc.).

2 Implémentation `Java`

Contraintes prélables

1. Le code rendu devra être "original": tout code, même partiel, provenant d'internet ou d'outil comme TchatGpt sera sanctionné par la note 0.
2. Aucune API, autre que celle fournit par `java.base`, ne devra être utilisée dans le cadre de ce projet.

Points évalués

1. Le nom de l'application rendue est `forms.jar`
2. L'application se lance avec `java -jar forms.jar`
3. L'application fonctionne correctement et il est possible de:
   (a) Créer un rectangle
   (b) Déplacer ce rectangle
   (c) Créer une forme comlexe (intersection de deux rectangles)
   (d) Déplacer la forme complexe
   (e) Sauvegarder/restaurer la forme complexe
   (f) Découvrir une fonctionalité "originale" (non spécifiée dans le cahier des charges)
4. L'extraction du contenu de l'archive avec `jar xf forms.jar` génère un répertoire/package `forms` contenant à la fois le code executable `*.class` et surtout le code source `*.java`
5. Il doit être possible de recompiler le code avec `javac -cp .  forms/*.java`
6. L'application doit pouvoir être lancée avec `java -cp .  forms.App`
7. Le code source (`*.java`) doit être correctement structuré (indentations, lisibilité, etc.)

8. L'extraction du contenu de l'archive devra aussi générer un fichier `Readme.txt` présentant: le(s) nom(s)/prénom(s) du/des auteur(s), une courte description de l'application (avec en particulier la fonctionnalité "originale" proposée).
   Ce fichier devra aussi préciser un exemple de "bug" possible ayant été pris en compte dans le code (ex. déplacement d'une forme en dehors de la fenêtre).

3 Documentation `UML`

Le logiciel proposé devra être construit de façon rigoureuse et documenté à l'aide des principaux diagrammes UML. En particulier, il est demandé un rapport intitulé "forms.pdf" composé des éléments suivants:

1. Un titre correspondant au nom du logiciel (ex. "EasyCSG").
2. Les nom/prénom des auteurs.
3. Une courte introduction présentant le contenu du rapport.
4. Une description des fonctionnalités réalisées (ou possibles) sur un diagramme des cas d'utilisation.
   NB. Pour l'ensemble des diagrammes, veillez à utiliser le maximum de concepts vus en cours ; par exemple, en faisant apparaître les différents types de relations entre Use Cases.
5. En utilisant un diagramme de séquence, donner les interactions nécessaires pour créer une différence de 2 rectangles qui est ensuite redimensionnée.
   NB. On veillera à faire apparaître les états du logiciel à chaque étape de la création et du redimensionnement.
6. Représenter le comportement (partiel) du logiciel sur un diagramme d'état (statechart). Ce diagramme devra faire apparaître: 1) les états définis à la question précédente, et 2) les transitions permettant de déplacer une forme.
7. Sur un diagramme de communication, illustrer les opérations réalisées pour redimensionner la forme de la Figure 2.

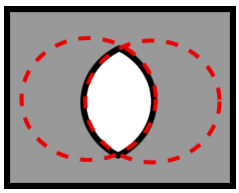

Figure 2: Exemple de forme.

8. Proposer un diagramme des classes détaillant les principaux éléments du logiciel réalisés.
   NB. On fera apparaître la/les classe(s) utilisée(s) pour permettre le rendu d'une forme sur l'interface graphique. On fera aussi apparaître les différents packages considérés.
9. Sur un diagramme de déploiement, expliquer comment se fait la sauvegarde d'une forme sur un serveur distant.

10. Enfin, proposer deux exemples de contraintes OCL pour un invariant et une pré-post condition d'opération.

11. Une courte conclusion résumant les possibilités, mais aussi les limites, du logiciel proposé avec quelques perspectives d'évolution possibles.

Le rapport rendu devant servir à évaluer votre connaissance du langage UML, privilégier la qualité des diagrammes (en utilisant le maximum d'éléments vus en cours - ex. faire apparaître les 2 types de relations possibles en Use Cases ou Packages) plutôt que la quantité (:il ne s'agit pas de rendre un rapport technique complet mais une ébauche illustrant quelques aspects importants du code).

4 Conclusion

- Les éléments attendus devront être rendus
  au plus tard le lundi 6 mai à 8:00

- Le code `Java` sera déposé à l'adresse:
  `https://java.laurent-thiry.fr/work/project`

- Le modèle `UML` sera déposé à l'adresse:
  `https://uml2.laurent-thiry.fr/work/project`

Attention: une seul dépot sera permis pour chacun des éléments précédents, et un seul membre par binome fera le dépot!