Tandian
Binta

Ce qui a été difficile :

- Les exceptions : J'ai eu du mal à trouver les bonnes exceptions, car toutes les classes se ressemblaient. On aurait pu répéter les mêmes exceptions à chaque fois, mais je ne l'ai pas fait, sinon vous auriez été mécontent. J'ai donc essayé de provoquer des exceptions dans mon code pour pouvoir les capturer et les inclure dans le code. J'ai utilisé le cours de qualité de développement pour obtenir certaines exceptions. Mais il était difficile de faire en sorte que le code provoque ces exceptions sans comprendre comment cela fonctionne.

- La méthode dessiner() : La méthode "dessiner" dans toutes les classes demandait une réflexion approfondie. Je n'avais pas compris comment cela fonctionnait, et c'est ma sœur qui m'a aidé à comprendre en faisant des dessins pour illustrer le raisonnement. Par exemple, un premier segment se dirige vers le deuxième, qui se dirige vers le troisième, le troisième vers le quatrième, qui à son tour se dirige vers le premier, formant ainsi une forme de type quadrilatère.

Ce qui a été fait :

- Les classes Quadrilatère, Chapeau, FormeComposee et Triangle ont été créer.
- Les méthodes nécessaires comme les constructeurs, les setter, les getter, les méthodes de dessin, de déplacement et la méthode pour récupère le type de forme ont été ajoutées à chaque classe.
- Les classes qui ont utiliser l'héritage de la classe Forme ont été définies et les méthodes abstrait de la mère ont été redéfini dans ses classe filles.
- J'ai ajouté la méthode toString() dans chaque classe afin d'améliorer ma compréhension du code et de répondre aux besoins des utilisateurs externes. Bien que cela n'ait pas été demandé, j'ai pensé que c'était nécessaire pour faciliter la lecture du code et fournir des informations claires sur les objets. J'ai également implémenté la méthode retireForme(), qui permet de supprimer une forme spécifique de la liste des formes. J'ai pris cette décision en considérant le fait que nous avions déjà la méthode ajouter pour ajouter des formes, et il était logique d'avoir une méthode équivalente pour supprimer facilement une forme sans supprimer la ligne de son code.

Ce qui n’a pas été fait :

- J'ai rencontré des difficultés pour ajouter les exceptions dans chaque classe à côté des méthodes. Chaque fois que j'essayais d'utiliser l'instruction "throw", cela générerait des erreurs et ne fonctionnait pas correctement. Par conséquent, j'ai décidé de gérer les exceptions dans les tests JUnit. J'ai également effectué des tests de qualité de développement sur JUnit pour vérifier si le code fonctionnait correctement, car il n'y avait pas beaucoup d'exceptions spécifiquement prévues dans ces classes. Donc on peut considère que les exception ont été fait ?

Ce qui fonctionne:

Tandian
Binta

- Les classes sont capables de créer des instances de formes géométriques avec les bon points.
- Les méthodes de dessin génèrent les segments nécessaires pour représenter les formes.
- Les méthodes déplacer permettent de déplacer les points des formes selon les coordonnées donnée.
- Les méthodes pour le type de la forme retournent les codes correspondants pour chaque classe.
- Les méthodes d'accès getter et de modification setter fonctionnent.

Ce qui ne fonctionne pas :

- Ce qui ne fonctionnait pas, c'était le dessin du rectangle car cela ne correspondait pas aux quatre points d'un quadrilatère. Il était nécessaire de calculer les coordonnées correctes pour chaque segment. Heureusement, le groupe de Yassine m'a aidé, car j'étais seul. Maintenant, cela fonctionne correctement.
- En ce qui concerne le test JUnit, il y a une assertion assertArrayEquals qui échoue car la méthode equals dans la classe Object compare les références des objets plutôt que leurs éléments. Je n'étais pas sûr si je devais redéfinir la méthode equals dans ma classe, donc j'ai décidé de laisser tomber cette idée. Je ne voulais pas ajouter de travail supplémentaire pour vous.

ce que je cherché:

- Thread.sleep(1000)