420-935-RO Concepts de la programmation orientée objet
TP2 – Casino (suite)

Sylvain Labranche — Collège de Rosemont

Été 2021

1 Énoncé

Votre Casino fonctionne à merveille et vous commencez à récolter les fruits de votre dur labeur. Comme tout individu au service du capitalisme, vous voulez étendre votre emprise et augmenter vos revenus. Vous décidez de scinder la population en plusieurs types de joueurs, d'offrir deux types de casinos ainsi que de nouveaux jeux.

Vous ne réécrirez pas tout le code à partir de 0, puisque vous maîtrisez maintenant l'héritage!

Votre programme a déjà les classes suivantes: Joueur, Casino, Jeu et TestCasino. Vous devez transformer Joueur, Casino et Jeu en des classes abstraites.

S'ajouteront les classes TestCasino2, JoueurPauvre, JoueurRiche, CasinoLegal, CasinoClandestin, PileOuFace ainsi qu'un autre jeu de votre choix.

Ces classes se retrouvent dans les paquetages Joueurs, Casinos, Jeux et Main.

Vous devez faire le diagramme UML de vos classes (pas besoin de refaire en détails Joueur, Casino et Jeu).

1. La classe Casino:
   - A un nouvel attribut: int capital. Le capital initial est de 100 000 $. Chaque fois qu'un joueur mise, le capital est augmenté. Chaque fois qu'un joueur gagne, le capital est diminué.
   - implémente l'interface **déjà existante** Comparable. Un casino est plus grand qu'un autre si la somme du capital des joueurs présents est plus grande que celle de l'autre.
   - implémente l'interface **que vous écrivez** ImpotsFonciers qui contient les méthodes suivantes:
     - payerImpôts() : implémentée dans chacun des Casinos.
     - evaluationMunicipale() : Un inspecteur vient évaluer la valeur totale du Casino : 1000$ multiplié par le nombre de joueurs présents dans le casino.
   - La classe Casino ne doit pas avoir deux tableaux (un de joueurs riches et un de joueurs pauvres): il doit seulement y avoir un tableau de Joueur (pensez au polymorphisme!).
   - Pour entrer dans un Casino, un joueurPauvre doit avoir au moins 10$, alors qu'un JoueurRiche doit avoir au moins 1000$. Cette vérification doit se faire dans la méthode *ajouterJoueur* de Casino.
2. La classe Joueur implémente l'interface Comparable. Un joueur est plus grand qu'un autre si son capital est plus grand.

   Si les deux joueurs ont le même capital, c'est l'ordre alphabétique de leur nom qui décidera.

3. CasinoLegal hérite de Casino et offre en plus les méthodes suivantes:

   - debutSpectacle() et finSpectacle() : Grégory Charles fait un spectacle au Casino! Personne ne ~~**veut**~~ **peut** manquer ça!

     Lorsque debutSpetacle() est appelée, aucun joueur ne peut jouer au Casino tant que finSpectacle() n'est pas appelée.
   - implémente l'interface ImpotsFonciers de la manière suivante:
     - payerImpôts() : Le Casino paie des impôts représentant 15 % de son capital.
   - Vous devez fournir au moins deux constructeurs, le constructeur par copie, les accesseurs et les mutateurs que vous jugez pertinents, la méthode toString et la méthode equals. **Vous ne devez pas nécessairement écrire toutes ces méthodes, pensez aux principes de l'héritage.**

4. CasinoClandestin hérite de Casino et offre en plus les méthodes suivantes:

   - descenteDePolice() : Tous les joueurs présents quittent le casino pour ne pas se faire arrêter.
   - implémente l'interface ImpotsFonciers de la manière suivante:
     - payerImpôts() : Le Casino ne paie rien. Il a 1 % de chance de voir 50% de son capital saisi par Revenu Québec.
   - Vous devez fournir au moins deux constructeurs, le constructeur par copie, les accesseurs et les mutateurs que vous jugez pertinents, la méthode toString et la méthode equals. **Vous ne devez pas nécessairement écrire toutes ces méthodes, pensez aux principes de l'héritage.**

5. JoueurPauvre hérite de Joueur et offre en plus les méthodes suivantes

   - collecterCheque() : **Si nous sommes le premier du mois**, le joueur peut collecter un chèque! Son capital augmente de 700$.
   - Vous devez fournir au moins deux constructeurs, le constructeur par copie, les accesseurs et les mutateurs que vous jugez pertinents, la méthode toString et la méthode equals. **Vous ne devez pas nécessairement écrire toutes ces méthodes, pensez aux principes de l'héritage.**

6. JoueurRiche hérite de Joueur et offre en plus plus les méthodes suivantes

   - banqueRoute() : Les investissements à l'étranger du joueur sont saisis par le gouvernement et ses comptes dans les paradis fiscaux sont fermés. Le capital tombe à 0 et le joueur quitte le casino, s'il est dans un casino.
   - Vous devez fournir au moins deux constructeurs, le constructeur par copie, les accesseurs et les mutateurs que vous jugez pertinents, la méthode toString et la méthode equals. **Vous ne devez pas nécessairement écrire toutes ces méthodes, pensez aux principes de l'héritage.**

7. Dans la classe Jeu, la méthode calculerGains(mise) doit être abstraite.

   - Votre jeu du TP1 hérite de Jeu. Le calcul des gains se fait de la même manière que dans le TP1.
   - Vous ajoutez le jeu Roulette. Un nombre aléatoire est tiré entre 0 et 36. Un joueur peut miser:

- Sur les rouges. En cas de gain, il double sa mise.
- Sur les noirs. En cas de gain, il double sa mise.
- Sur un numéro de 1 à 36. En cas de gain, il remporte 36 fois sa mise.
- Aucun gain si le 0 est tiré.

• La méthode toString() affiche les règles du jeu.

8. La classe TestCasino2 aura une méthode main qui agit comme programme de test. Vous êtes responsable d'y tester chacun des ajouts au Casino par rapport au TP1. Rappelez vous que la classe de test est aussi évaluée : je vérifie si vous avez pensé à tout tester!

2 Méthodologie proposée

1. D'un point de vue pédagogique, il est préférable pour vous de faire le travail à partir de votre propre TP1. Cependant, si vous n'êtes pas satisfait de votre TP1, vous pouvez utiliser le TP1 que je fournis.
2. Commencez par le diagramme UML. Vous aurez une meilleure idée de l'organisation de vos classes.
3. Allez-y étape par étape et testez à chaque fois. Implémentez une méthode et testez-la immédiatement après. Mettez le code de tests des méthodes fonctionnelles en commentaire dans votre programme principal.
4. Commencez par les méthodes les plus faciles (constructeurs, toString, accesseurs, mutateurs, etc.) et allez-y une à la fois.

Pour obtenir un nombre aléatoire:
La méthode Math.Random() retourne un nombre aléatoire entre 0 et 1 (1 n'est pas inclus).
Pour obtenir un nombre aléatoire entre 1 et n inclusivement, on fait :
$(int)(n * Math.Random()) + 1;$
Pour obtenir un nombre aléatoire qui est soit 0, soit 1, on fait :
$(int)(2 * Math.random());$

3 Modalités

Ce travail est fait individuellement. Vous pouvez collaborer, mais vous **devez** écrire votre propre code. Un plagiat, même partiel, du code d'un autre étudiant entraînera la note 0 pour les deux étudiants.

4 Remise

La date limite pour la remise est prévue **le dimanche 26 septembre 2021 à 23h59**. Vous aurez plusieurs heures en classe pour travailler sur votre TP.

Vous remettez le code dans une archive .Zip dans la boîte prévue à cet effet sur Léa.
Tout retard entraînera une note de zéro.

5 Évaluation

Ce travail compte pour 30 % de votre session.

Le diagramme UML compte pour 6 %, les classes Joueur, Casino et Jeu pour 1 %, chacune des classes CasinoLegal, CasinoClandestin, JoueurRiche, JoueurPauvre et Roulette pour 3% et votre programme de test pour 6 %. La qualité du code, l'implémentation correcte des méthodes et l'exhaustivité des tests seront évalués.