TRAVAIL PRATIQUE 1
LES TOURS DE HANOI

Pondération : 15 %
Équipe de 2 (obligatoire)

OBJECTIF PÉDAGOGIQUE

Développer en programmation objet une application simple (5 ou 6 classes principales) utilisant une classe collection.

L'atteinte de cet objectif exige de:

- maîtriser la syntaxe Java;
- maîtriser les classes, les objets, l'héritage et les associations entre classes;
- connaître les éléments de base nécessaires à la construction d'une interface graphique Windows en Java (JFrame, JPanel, JButton, …);
- connaître les méthodes événementielles pour compléter le code de gestion des événements;
- savoir dessiner sur un panneau (méthode paintComponent()).

MANDAT

Concevoir, spécifier, réaliser, documenter et tester les classes de base nécessaires au développement en programmation objet d'une application Java permettant de jouer aux « Tours de Hanoi ».

PRÉSENTATION DU JEU LES TOURS DE HANOI

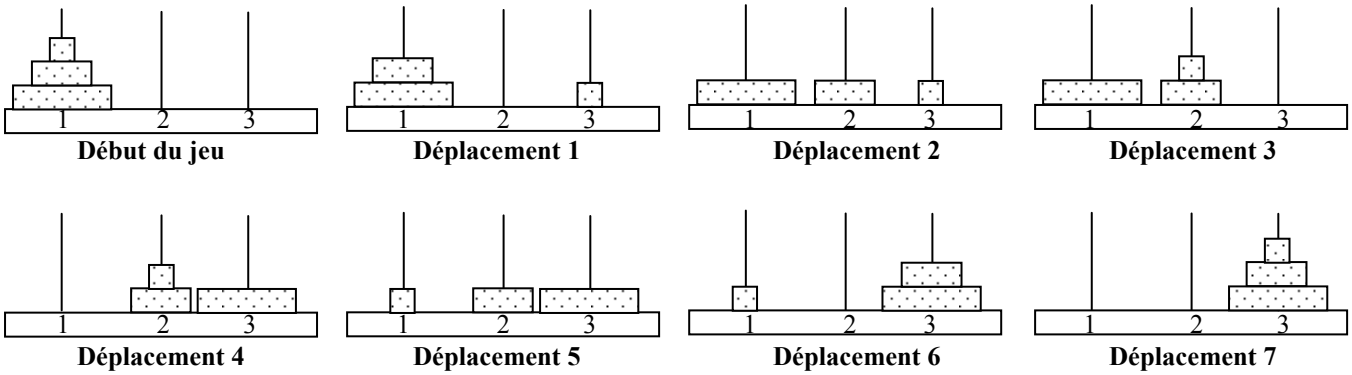
Le jeu physique des tours de Hanoi est constitué de :

- 3 tours (3 tiges de bois) numérotées de 1 à 3;
- n disques (n <= 64), tous d'un diamètre différent et percés en leur centre (pour pouvoir s'enfiler sur les tiges de bois).

Au départ, les disques sont tous empilés sur une tour. Le but du jeu est de transporter tous les disques sur une tour d'arrivée, différente de la tour de départ, en utilisant la troisième tour pour des déplacements intermédiaires et en respectant les règles suivantes :

- ne déplacer qu'un disque à la fois;
- ne déplacer que le disque du dessus d'une tour;
- ne jamais placer un disque sur un autre qui soit de plus petit diamètre.

La figure suivante illustre la solution du jeu avec 3 disques (n = 3). Pour clarifier l'exemple, les tours sont numérotées de 1 à 3, la tour 1 représentant la tour de départ, la tour 3 la tour d'arrivée et la tour 2 la tour intermédiaire.



SPÉCIFICATIONS DE L'APPLICATION

Fonctionnalités

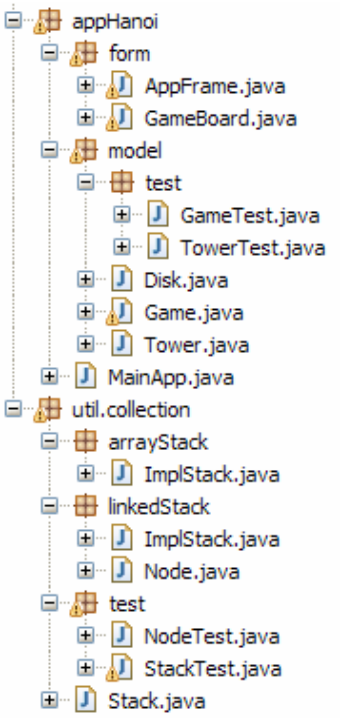
Vous avez à réaliser une application permettant de jouer de manière interactive au jeu des Tours de Hanoi avec 3, 4 ou 5 disques. Votre application doit supporter les fonctionnalités minimales suivantes :

- Au lancement de l'application :
  - Exécuter une suite de tests JUnit afin d'exécuter les classes de tests de l'application;
  - Puis accéder au jeu des Tours de Hanoi.

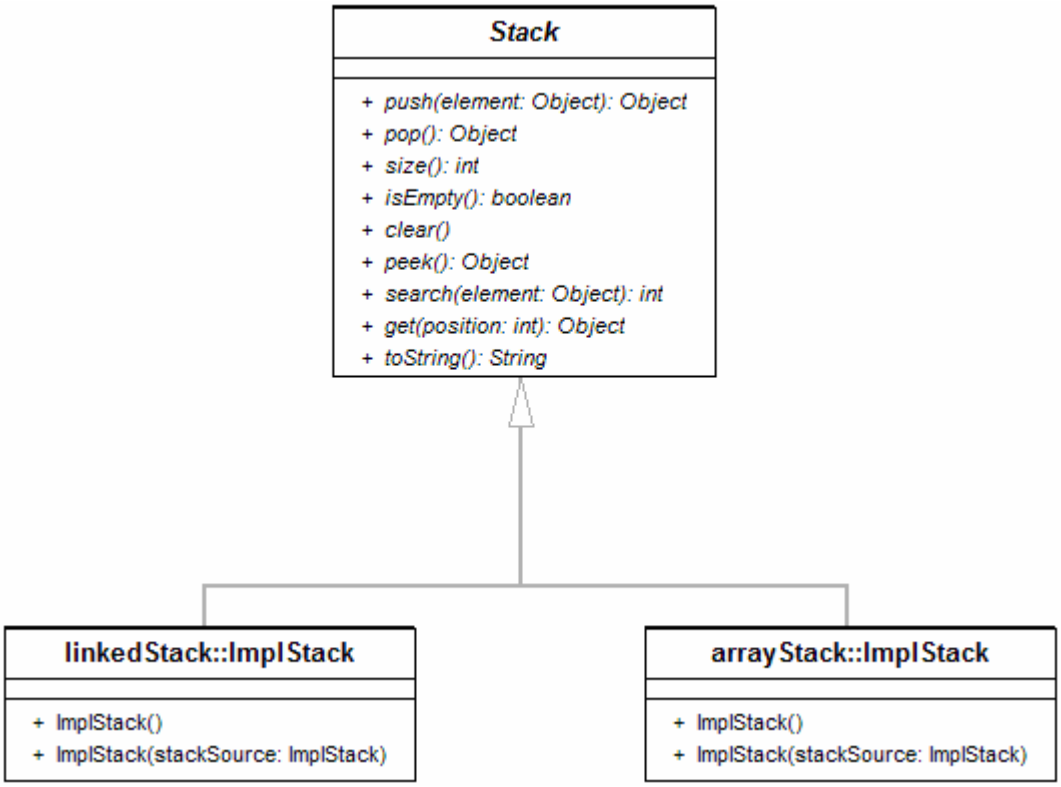
- Les classes tests Junit sont exécutées dans l'ordre suivant :
  - util.collection.test.NodeTest : test classe nœud (util.collection.linkedStack.Node)
  - util.collection.test.StackTest : test classe pile (util.collection.linkedStack.ImplStack ou util.collection.arrayStack.ImplStack)
  - appHanoi.model.TowerTest : test classe tour (appHanoi.model.Tower)
  - appHanoi.model.GameTest : test classe jeu (appHanoi.model.Game)

- Le mode jeu propose une interface utilisateur graphique par système de fenêtrage simple dans laquelle :
  - **le titre de la fenêtre indique le nom des 2 étudiants programmeurs**;
  - le joueur peut commencer une partie : il indique le nombre de disques (3, 4 ou 5);
  - le joueur doit être capable de spécifier les déplacements des disques;
  - après chaque déplacement, l'état des trois tours doit être affiché;
  - l'application doit avertir le joueur lorsque la partie est gagnée.

Classes principales



Classe abstraite Stack (Pile) : *util.collection.Stack*
C'est une classe collection de type premier entré, premier sorti (LIFO) qui permet de maintenir une pile d'objets. Cette classe abstraite contient les méthodes usuelles de cette collection de données classique :



La documentation javadoc (en anglais) des méthodes de la classe Stack vous sera fournie et doit être respectée.

2 Classes ImplStack : *util.collection.arrayStack* et *util.collection.linkedStack*
La classe Stack doit avoir 2 sous-classes concrètes qui implémentent la classe abstraite Stack :

- une classe utilisant un tableau : ***util.collection.arrayStack.ImplStack***;
- une classe utilisant une liste simplement chaînée : ***util.collection.linkedStack.ImplStack***, ***util.collection.linkedStack.Node***.

Ces 2 classes auront exactement les mêmes membres publics (voir UML Stack), et la même documentation javadoc. Seules les propriétés privées, le code des méthodes et les commentaires diffèrent.
On devrait donc pouvoir utiliser indifféremment l'une ou l'autre : il suffit de modifier les instructions d'import de la classe ImplStack pour utiliser l'un ou l'autre des implémentations.

- Les classes de l'application (Tower et StackTest) devront fonctionner en utilisant n'importe laquelle des 2 classes ImplStack (array ou linked list).

Vous démontrez ici le concept d'encapsulation : même si la réalisation d'une classe change, si l'interface de la classe (la signature et le comportement des méthodes) reste identique, le code de l'application (le code des classes utilisatrices) n'a pas à être modifié.

Classe Disk (Disque) : *appHanoi.model.Disk*
Cette classe, très simple, représente un disque du jeu de Hanoi: un disque est défini par un diamètre (nombre entier), ou un numéro proportionnel à son diamètre, et une couleur. La classe aura les membres que vous jugerez bon d'inclure

Classe Tower (Tour) : *appHanoi.model.Tower*
Cette classe représente une tour du jeu de Hanoi. Les membres de la classe doivent permettre de décrire et de gérer l'état d'une tour, c'est-à-dire l'empilement des disques sur une tour. La classe aura les membres que vous jugerez bon d'inclure.
Contrainte : Cette classe est une sous-classe d'une classe ImplStack.

Classe Game (Jeu) : *appHanoi.model.Game*
Cette classe permet de garder la trace du statut des trois tours au cours d'une partie et a la responsabilité de gérer les déplacements des disques entre les tours. Elle s'occupe également de l'affichage des tours. La classe aura les membres que vous jugerez bon d'inclure.

Classe GameBoard (PlateauJeu) : *appHanoi.form.GameBoard*
Cette classe a la responsabilité de gérer l'interface graphique et les événements déclenchés par l'interaction avec l'utilisateur qui joue aux Tours de Hanoi. La classe aura les membres que vous jugerez bon d'inclure. Toutefois, excepté les constructeurs et les méthodes événementielles, cette classe ne devrait contenir que des méthodes privées. Une version élémentaire de cette classe vous sera fournie en classe.

Classe AppFrame (AppCadre) : *appHanoi.form.AppFrame*
Cette classe représente le cadre principal de l'application. Elle permet d'accéder au jeu des Tours de Hanoi. La classe aura les membres que vous jugerez bon d'inclure. Toutefois, excepté le constructeur, cette classe ne devrait contenir que des méthodes privées.

Diagramme des classes UML

À prime abord, pour vous assurer de la qualité de votre application, vous devez concevoir toutes les classes requises pour votre application selon la notation UML.

Donc, pour chacune des classes :

- Identifier clairement son nom;
- Identifier les relations entre les classes : héritage et associations;
- Identifier toutes les propriétés privées et publiques utilisées;
- Identifier toutes les méthodes privées et publiques utilisées;

Une version élémentaire et préliminaire du diagramme de classes sera réalisée en classe.

Avant la remise finale du TP1, vous aurez à remettre à votre professeur la version du diagramme de classes réalisée lors de l'étape de conception.

À la remise finale du TP1, vous aurez à remettre dans le dossier de programmation le diagramme de classes final, complet (membres publics et privés), documentant et représentant fidèlement les classes que vous aurez finalement réalisées.

ASSURANCE QUALITÉ – EXIGENCES

Classes de tests

- Tests des classes ***Node, Stack (ImplStack), Tower, Game*** : pour chacune de ces classes, vous devez réaliser une classe de test JUnit permettant de valider tous les constructeurs et méthodes publics de la classe correspondante.
- Respecter le standard des classes de test : une méthode de test pour chaque méthode publique de la classe testée.
- Identifier clairement dans le code les différents cas testés; soyez clairs et précis;
- Assurez-vous de tester le respect et le non-respect des contraintes et validations de chaque méthode (pré et post conditions);
- N'oubliez pas de tester les paramètres;
- Pensez aux valeurs et situations limites;
- On apportera une attention particulière aux tests d'une **PILE VIDE**.

Scénarios

- Pour vous assurer de la qualité de votre application, vous devez concevoir et réaliser des scénarios de l'application;
- Vous devez produire, sous la forme d'un tableau, une liste de contrôle des scénarios garantissant l'assurance qualité de votre application;

- Les scénarios permettent de tester l'interface graphique de votre application et l'interaction avec l'utilisateur : ils complètent l'assurance qualité réalisée par les classes de tests;
- Les scénarios de tests doivent être regroupés par catégorie : identifiez clairement les sections et les sous-sections;
- Pensez aux situations limites;
- La description de chaque scénario doit être succincte mais claire et précise. Elle doit inclure :
    - une description de la situation testée;
    - une description des résultats attendus;
    - une description du résultat obtenu.

TECHNIQUES DE DESIGN ET DE PROGRAMMATION - CONTRAINTES

- Assurez-vous que vos classes disposent des validations et des méthodes nécessaires pour garantir le respect des règles et contraintes du jeu;
  N'oubliez pas que chaque classe a la responsabilité de maintenir ses objets dans un état valide;
- Respectez les standards et règles de nomenclature et de programmation définis en classe;
- Utilisez une seule langue (français ou anglais) dans votre code;
- La documentation doit être en français;
- Découpez votre code (n'hésitez pas à utiliser des méthodes privées à cet effet);
- Utilisez systématiquement des propriétés statiques et finales pour les constantes littérales;
- Commentez les propriétés privées et les méthodes privées;
- Commentez votre code;
- L'interface de l'application est laissée à votre discrétion; cependant l'interface doit être en français;
- Assurez-vous qu'il n'y ait aucune erreur et aucun warning à la compilation;
  L'erreur d'exécution (« NumberFormatException »), provoquée lors d'une tentative de conversion d'une chaîne de caractères non numérique en entier (Integer), est acceptée.

DOCUMENTATION JAVADOC -EXIGENCES

Toutes les classes de votre application doivent être documentées selon le standard Javadoc.

- Utilisez les balises : @param, @return, @see;
- Documentez les responsabilités générales, le rôle de chaque classe. Soyez clairs et précis;
- Documentez, chaque membre public de vos classes : décrivez son rôle, précisez les contraintes à respecter, spécifier les validations. Soyez clairs et précis;
- Documentez en français;
- La documentation Javadoc html doit être générée pour tous les paquets, toutes les classes et tous les membres publics des classes de votre application.

Français

- Toute faute de français dans l'interface utilisateur de l'application est pénalisée d'un (1) point.
- Conformément à la politique du Cégep concernant la qualité du français écrit, une pénalité de 10% peut en plus être attribuée selon la qualité du français utilisé dans le code (commentaires et documentation) et dans le dossier de programmation.

ÉVALUATION

Se référer à la section « Évaluations »-« Critères d'évaluation des travaux » du plan de cours pour plus de détails.

Cotation

Cote	Note
A+, A, A-	100, 95, 90
B+, B, B-	85, 80, 75
C+, C, C-	70, 65, 60
D	50
E	30
Z	0

Pondération

1. Réalisation complète de classes collections « génériques »	15%
2. Conception et réalisation des classes (incluant UML)	10%
3. Classes de tests	10%
4. Scénarios (complément aux classes de tests)	10%
5. Documentation Javadoc des classes	10%
6. Qualité du code : documentation, structure, découpage, standards	15%
7. Professionnalisme du travail et du dossier	10%
8. Réalisation des fonctionnalités de l'application	10%
9. Qualité et convivialité de l'interface	10%

À noter :
Une application opérationnelle ne garantit pas la cote de passage. L'architecture objet et la qualité du code réalisé sont primordiales. La responsabilité de chaque classe doit être bien définie. Le code doit être facile à lire, clair, découpé, bien structuré, commenté et aisé à maintenir. Le dossier de documentation doit être complet et de qualité professionnelle.

Si le professeur le juge à propos, tout étudiant(e) pourra être convoqué(e) à une rencontre d'évaluation pour vérifier son degré d'acquisition des connaissances et d'appréhension de la solution proposée.

FONCTIONNALITÉS ET RÉALISATIONS ADDITIONNELLES - BONUS

Toute fonctionnalité ou réalisation additionnelle à l'application est susceptible d'obtenir des points en bonus, pour un maximum de 10 points de bonus et de 100 points de la note finale.

Toutefois, ces fonctionnalités ne seront considérées que si les spécifications de l'application ont été respectées et réalisées selon les critères de qualité exigés; l'évaluation du travail devra au préalable obtenir une cote supérieure ou égale à B.

Dans votre contrat de livraison, vous devez clairement identifier toutes les sources de documentation et d'information utilisées pour développer ces fonctionnalités additionnelles : aide en ligne d'Eclipse, livres, sites Internet …

Suggestions (la pondération est donnée à titre indicatif) :

- gérer le nom et le score du joueur; [*max 3 pts*]
- proposer une résolution automatique du jeu des Tours de Hanoi; [*max 5 pts*]
- réalisation d'une classe collection Pile la plus complète et la plus réutilisable possible : ajout de méthodes utiles et classe générique; [*max 5 pts*]
- permettre de choisir l'implémentation de Stack à utiliser à l'exécution (création dynamique d'objet de sous-classes). [*max 5 pts*]

DOCUMENTS À RÉALISER ET À REMETTRE

Vous devez remettre un dossier de documentation de ***qualité professionnelle***.

Le dossier doit être présenté dans un cartable identifié et doit contenir, dans l'ordre, les documents énumérés ci-dessous. Chaque section et chaque classe doit être clairement identifiée : vous devez utiliser des séparateurs.

√ *Contrat de livraison :*

Ce document :

- d'une part certifie que le travail a été réalisé selon les normes, standards et contraintes exigés,
- d'autre part identifie les fonctionnalités opérationnelles ou non opérationnelles de votre système selon les critères de qualité requis.

Vous devez également préciser si l'application a été testée avec les 2 classes ImplStack.

Énumérez enfin les fonctionnalités additionnelles disponibles, ainsi que les sources de documentation ou d'information utilisées.

Ce document doit refléter l'état réel de votre application et de l'assurance qualité réalisée, en conformité avec la liste de contrôle des scénarios d'utilisation. Le non-respect de cette contrainte pourrait entraîner une pénalité allant jusqu'au refus d'accepter le travail.

√ *Auto-évaluation :*

Insérer ici votre document d'auto-évaluation d'équipe complété :

- évaluez chaque critère (cote de A à Z),
- complétez la « check-list »,
- commentez au besoin.

√ *Diagramme de classes UML :*

Ce document présente graphiquement selon les normes vues en classe les classes (propriétés et méthodes) ainsi que les relations d'héritage et d'association existantes entre chacune d'elles.

√ *Liste des scénarios:*

Ce document décrit les scénarios à réaliser pour assurer la qualité de l'application (voir section « Assurance qualité – exigences »-« Scénarios »).

√ *Interface :*

Ce document contient une image de toutes les fenêtres de votre application : fenêtre principale et boites de dialogues (message et saisie).

√ ***Dossier de programmation :***
Ce document contient le code des fichiers .java de toutes les classes de votre application (imprimé à partir du logiciel de développement).

Identifiez chaque classe par un ***séparateur nommé*** dans le cartable (la classe de tests suivant sa classe associée). Le dossier sera **non corrigé** advenant le non respect de ce standard.

Assurez-vous de la clarté du code imprimé (retour à la ligne).

√ ***Planification du projet :***
La planification initiale de développement doit être inclue dans votre dossier de programmation. Activités : conception des classes, programmation des classes, documentation des classes, scénarios d'utilisation, déboguer l'application, dossier de documentation.

√ ***Feuille de temps :***
Chaque étudiant doit maintenir un rapport quotidien et individuel des activités réalisées pour le travail pratique.

√ ***L'application :***
Le nom de votre projet doit inclure le nom des étudiants : « TP1-NomEquipier1-NomEquipier2 ». Copier le .zip du projet complet de votre application sur la plateforme LÉA dans le répertoire identifié à cette fin.

DATE DE REMISE

jeudi 23 septembre 2010 – 16 h

Pour plus d'information sur les tours de Hanoi, pour des exemples de jeux :

Tour de Hanoi : jeu, légende, analyse, solution :
http://perso.wanadoo.fr/therese.eveilleau/pages/jeux_mat/textes/tour_hanoi.htm
Tour de Hanoi en Java : http://www.mazeworks.com/hanoi/index.htm
Tours de Hanoi : wikipédia http://fr.wikipedia.org/wiki/Tours_de_Hano%C3%AF
Je de Hanoi : une interface utilisateur simple http://pagesperso-orange.fr/jeux.lulu/html/hanoi/hanoi1.htm