TP1

Tous les exercices sont rangés dans des packages nommés exo1, exo2, exo3, …
Le dossier src/ contient le code des exercices à réaliser.
Le dossier correction/ contient la correction de ces exercices.
Le dossier test/ contient les tests unitaires avec JUnit réalisés pour tester ce que vous faites. Dans le cas de ce TP, il n'y a que des tests pour le premier exercice.

Exercice 1 : Maths

L'objectif de cet exercice est de réaliser quelques fonctions mathématiques standards.

Le code des méthodes à implémenter se trouve dans la classe Maths située dans le répertoire "src/exo1/math" du projet TP1.

Les fonctions à implémenter sont :

1. *static public int fact(int n)* -> Renvoi la factorielle du nombre n en utilisant des appels récursifs. Donc la fonction doit s'appeler elle-même pour réaliser le traitement.
2. *static public int factIteratif(int n)* -> Renvoi la factorielle du nombre n de manière itérative. Cela implique de ne pas utiliser d'appels récursifs, mais d'utiliser une boucle (for, while, …). Chaque itération de la boucle est l'équivalent d'un appel récursif et le traitement à réaliser (multiplication dans le cas de la factorielle) est répétée à chaque itération.
3. *static public int pow(int n, int d)* -> Renvoi la puissance du nombre n au degré d. Cela implique qu'il faut multiplier d fois le nombre n. Donc $3^4 = 3 * 3 * 3 * 3 = 81$. Cette fonction doit être réalisée de manière récursive.
4. *static public int pow(int n, int d)* -> Renvoi la puissance du nombre n au degré d, mais de manière itérative (donc avec une boucle for / while).
5. *static public int abs (int n)* -> Renvoi la valeur absolue du nombre n.
6. *static public int min (int a, int b)* -> Renvoi le minimum entre a et b, soit le nombre le plus petit des deux.
7. *static public int max (int a, int b)* -> Renvoi le maximum entre a et b, soit le nombre le plus grand des deux.

Afin de tester le résultat de chaque fonction, la classe MathsTest contient un ensemble de tests avec JUnit. Il faut donc lancer l'ensemble des tests (Clic droit -> Run As -> JUnit Test). La classe se situe dans le dossier test, dans le package exo1.math

Exercice 2 : AppStore

Le monde du mobile est en plein essor. Tous les fournisseurs de systèmes (Apple, Windows, Google, …) ont créé un “magasin d’applications” comme l’AppStore, ou Google Play. Nous allons donc lancer notre propre magasin d’applications.

I°) La classe Application

Puisque nous créons un store d’applications, il faut commencer par créer une classe Application. Cette classe est la représentation d’une application stockée dans notre store et se veut composer d’un identifiant unique, d’un nom, d’une description et d’une popularité.

Créer une classe *Application* ayant pour variables d’instance un entier id, un entier popularite, une chaîne de caractères name et une chaîne de caractères description.

Créer une variable static compteur, initialisé à 1.

Créer un constructeur prenant en paramètre un nom et une description. Ce constructeur initialise les variables d’instance nom / description par le biais des paramètres décrits précédemment, met la popularite à 0 et met l’identifiant de l’application (le champs id) a la valeur du compteur. Ce-dernier est incrémenté par la suite, afin d’obtenir un id toujours différent (de la même forme que l’auto increment des bases de données).

Créer des méthodes d’accès et de modification de ces variables d’instance. Cela implique d’avoir une méthode qui retourne simplement l’attribut (un getter) et une méthode qui prend en paramètre la nouvelle valeur de l’attribut et modifie la variable associée (setter). Par exemple vous créerez la méthode “*public void setName(String name)*” qui modifiera la variable d’instance name pour la remplacer par le paramètre.
Attention, ne pas créer de setter pour les attributs popularite et id.

Créer une méthode “*public void incrementPopularite()*” qui incrémente la popularité de l’application de 1.

Redéfinissez la méthode “*public boolean equals(Object ob)*”. On considère que deux applications sont équivalentes si leurs identifiants sont les mêmes.

Redéfinissez la méthode “*public String toString()*” qui renvoie simplement le nom de l’application.

II°) La classe AppStore

Maintenant que nous avons nos applications, nous allons devoir les stocker et proposer un ensemble de services d’ajout / suppression / recherche pour les particuliers.

Créer la classe AppStore contenant comme variables d’instance un tableau de Application (Application[] tab) et un entier servant de compteur d’applications. A chaque ajout d’une application, nous incrémenterons le compteur, et le décrémenterons lors de la suppression d’une application. Ce compteur représente donc le nombre d’applications du store.

Créer un constructeur sans paramètres qui initialise le tableau avec une taille de 200 et qui met le compteur

à 0.

Créer une méthode "*public boolean isFull()*" qui renvoie vrai si le store est plein, donc que l'on ajouté 200 applications au store.

Créer une méthode "*public int size()*" qui renvoie le nombre d'applications actuellement dans le store. **Attention cette méthode ne renvoie pas la taille du tableau, qui est de 200, mais le nombre d'applications contenues dans le tableau.**

Créer une méthode "*public boolean addApplication(Application a)*" qui ajoute une application dans le store. Si le store est plein, cette méthode renvoie simplement false. Sinon elle ajoute l'application et renvoie vrai. L'insertion fonctionne de manière identique à l'insertion dans une pile.

Créer une méthode "public int indexOfApplication(Application a)" qui renvoie la position de l'application spécifiée dans le store. Ainsi si l'application est recherchée est le 3ème élément du store, elle renvoie 3. Si l'application n'est pas trouvée, cette méthode renvoie -1.

Créer une méthode "public Application find(String name)" qui renvoie la première application dont le nom est spécifié. Si l'application n'existe pas, cette méthode renvoie null.

Créer une méthode "public Application find(int id)" qui renvoie la première application dont l'identifiant est spécifié. Si l'application n'existe pas, cette méthode renvoie null.

Créer une méthode "public boolean removeApplication(Application a)" qui supprime l'application spécifiée. Pensez à utilier des méthodes déjà existantes pour retrouver l'application à supprimer. Si l'application a supprimé n'existe pas, cette méthode renvoie false. Puisque l'insertion se fait à la manière d'une pile, nous ne pouvons pas laisser de "trous" dans le tableau. Par exemple si nous avons 5 applications, qui sont donc respectivement dans les 5 premières entrées du tableau et que nous voulons supprimer la 3ème, il faut penser à mettre les applications 4 et 5 aux positions 3 et 4.

Ce qui donne :

App 1
App 2
App 3
App 4
App 5

Suppression de l'application 3

App 1
App 2

App 4
App 5

Nous avons bien décalé toutes les applications après celle à supprimer pour laisser la fin vide, et sans mettre de trous.

Exercice 3 : Course de voitures

Si vous êtes arrivés ici, c'est que vous pouvez commencer à faire du Java dans la vie active ! C'est pour cela que l'entreprise éditrice du jeu mobile CrazyTrack vous demande de développer un module pour leur application mobile. CrazyTrack est un jeu de courses de voitures pour téléphone Android. Il possède donc déjà du code que vous ne pouvez pas modifier puisque réaliser par l'équipe de l'entreprise.

Chaque circuit de l'application est décrit dans un fichier, que vous trouverez dans le dossier track/, à la racine du projet Eclipse.

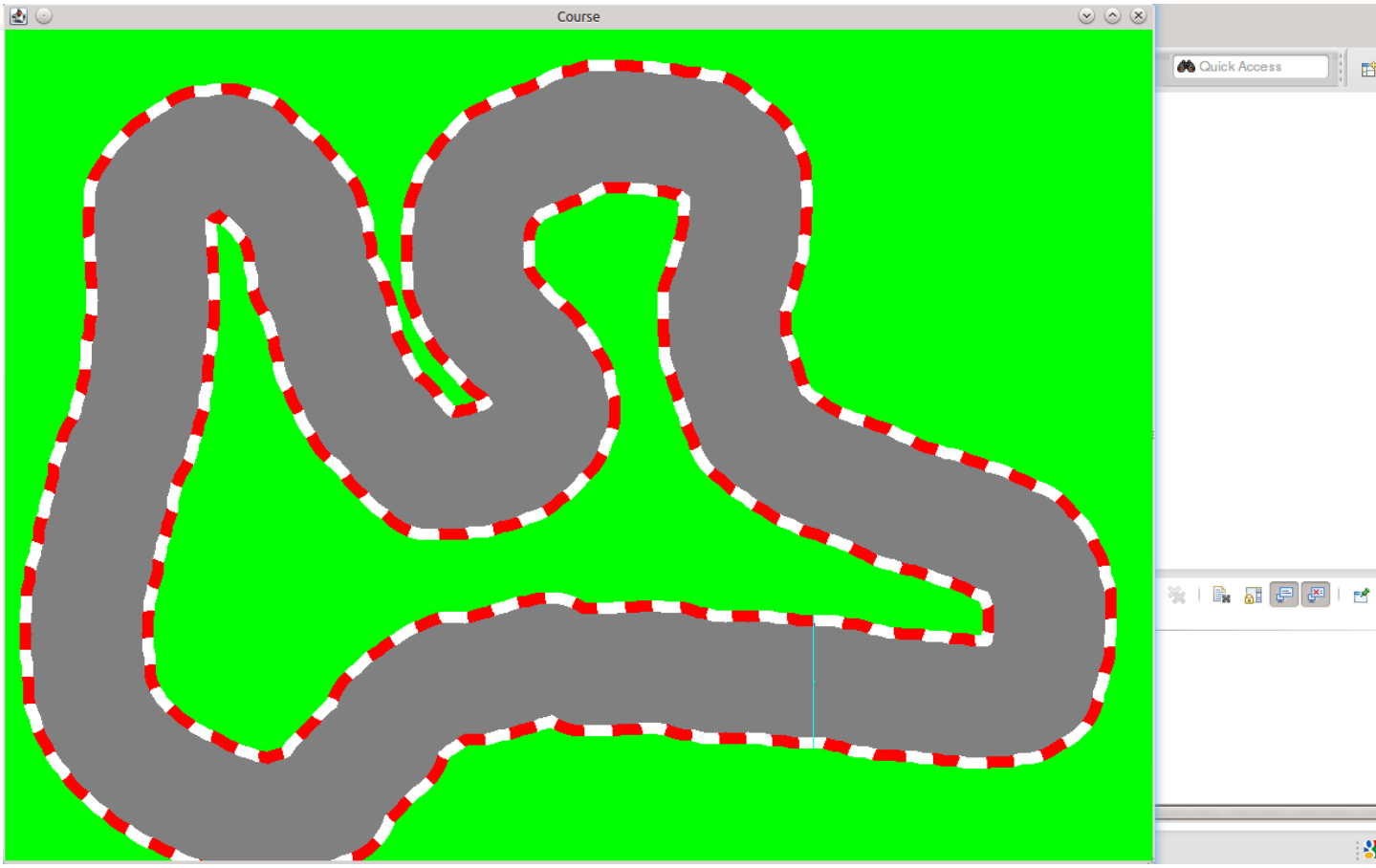
Tous les fichiers ont le format suivant :

1. longueur
2. hauteur
3. contenu

Par exemple :
1024
768
gggggggggggggggggggggggggggggggggggggggggggggggvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
gggggggggggggggggggggggggggggggggggggggggggggggvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

Votre tâche consiste à récupérer ce fichier et à générer une image depuis les données contenues. Chaque lettre contenues sous la taille (1024 et 768) correspond à un pixel de l'image. Ainsi dans le cas précédent il y a 768 lignes contenant chacune 1024 caractères. On obtient donc une image de 1024x768 pixels.

Voici le résultat obtenu avec le fichier "1_safe.trk", situé dans le dossier track.



La partie graphique est déjà existante via les classes TrackPanel et TrackFrame, vous n'avez donc pas à toucher à ces classes. De plus, une classe TrackReader a été réalisée pour la lecture dans le fichier. Cette-dernière permet de renvoyer les lignes du fichier dans un tableau. Ainsi, dans le cas précédent, on se retrouve avec un tableau de String[] contenant 770 lignes (les deux premières lignes étant les dimensions).

Le main de l'application est dans la classe TrackFrame, vous devez donc lancer l'application depuis celle-ci. La première ligne du main contient le chemin vers le fichier dont on veut afficher le circuit. Changez sa valeur si vous désirez voir d'autres circuits.

Votre travail se situe dans la classe ImageBuilder du package exo3.image et l'implémentation des méthodes "*public void read(String filename)*" et "*private Color conversion(char carac)*". Fort heureusement, un gentil ingénieur a documenté la description de l'algorithme à implémenter. Vous trouverez cette documentation directement au-dessus des méthodes à réaliser, dans la classe ImageBuilder.