Remarque préliminaire : les questions contenues dans cet énoncé appellent des réponses sous différentes formes : écriture de code Java, explications textuelles, etc. Le rendu final prendra la forme d'un unique rapport répondant aux différentes questions (numérotées), sous forme PDF. Les codes sources écrits en réponse à chacune des questions seront insérés dans le rapport aux emplacements appropriés. Le rapport contiendra également en annexe les codes sources complets. Une attention particulière devra être apportée à la forme du rapport (en plus, bien entendu, du fond).

A. Exceptions : première expérience

1. Ecrire une classe Tableau qui dispose de deux attributs : une longueur (entier) et un tableau de valeurs (tableau d'entiers). La classe devra offrir un constructeur prenant en paramètre le nombre de cases du tableau, ainsi que des accesseurs en lecture / écriture (prenant en paramètre l'indice de la case dans laquelle lire ou écrire).

2. Ecrire un programme de test de la classe précédente, sous format d'une méthode main dans une classe TestTableau. Que se passe-t'il en cas d'accès à :
   - une case d'indice négatif ?
   - une case d'indice supérieur à la taille du tableau ?
   - une case sans avoir initialisé le tableau ?

   L'erreur rencontrée lors de l'exécution du programme se traduit par la génération d'une exception. Ce mécanisme offert par Java permet de faciliter le débogage, et surtout d'anticiper les erreurs pouvant survenir et ainsi de prédéfinir les réactions du programme face à ces erreurs.

3. Le message d'erreur produit par la JVM fournit plusieurs informations. Identifier le type d'erreur, et localiser l'erreur (classe, méthode, ligne). Pourquoi plusieurs classes/méthodes/lignes sont-elles indiquées par Java ?

B. Instructions try/catch et throws

1. Ajouter un constructeur Tableau à la classe précédente, qui permet de renseigner un tableau en lisant des nombres présents dans un fichier texte (on supposera que le premier nombre du fichier donnera le nombre d'éléments à lire, et sera suivi par les différentes valeurs du tableau).

2. Sans gestion spécifique des IOException, Java autorise-t'il à compiler la classe ? Pourquoi ?

3. Apporter les modifications nécessaires au travers d'un bloc try / catch. En consultant la Javadoc, est-il possible de savoir quelles sont les méthodes qui nécessitent un traitement particulier des exceptions ?

4. Modifier le programme de test de la classe précédente pour faire appel au nouveau constructeur. Que se passe-t'il si le nom de fichier passé en paramètre est erroné (par exemple un fichier inexistant) : à la compilation ? à l'exécution ?

   Dans certains cas, il n'est pas possible de déterminer localement comment traiter une exception. Il est alors préférable de transférer la gestion de cette exception à la méthode appelante. On utilise pour ce faire le mot-clé throws dans la signature de la méthode.

5. Modifier le constructeur de la classe Tableau pour éviter d'avoir à y gérer l'exception au travers d'un bloc try / catch. Quelles sont les autres modifications à apporter à l'ensemble des classes pour pouvoir exécuter à nouveau le programme de test ?

6. Dans la classe de Test, proposer différentes manières de gérer l'exception.

7. Quelle est la principale différence entre les exceptions rencontrées ici et celles du premier exercice ?
   (conseil : lire la Javadoc de la classe RuntimeException)

C. Génération d'exceptions par le programmeur

Dans certains cas, il peut être utile pour un programmeur de décider de la génération d'une exception. Pour cela, on utilise l'instruction throw instanceException, où instanceException est un objet instance d'Exception (ou d'une sous-classe).

1. Modifier le code de la classe Tableau pour générer une exception lors de l'accès à une case d'indice erroné (négatif ou supérieur à la taille du tableau), en précisant la raison de l'erreur à l'utilisateur. On utilisera pour cela le paramètre String du constructeur de la classe Exception.

2. La solution précédente n'est pas très élégante. En effet, elle ne permet pas de distinguer les erreurs d'accès aux cases du tableau de celles d'accès au fichier. Il est préférable d'écrire une nouvelle classe d'erreur (par exemple ici : TableauException). Apporter les modifications nécessaires à l'ensemble des classes pour suivre cette recommandation. On fera la gestion des exceptions par des instructions try / catch uniquement dans le main de la classe TestTableau. Que se passe-t'il si on ne gère pas ces exceptions dans le main (mais qu'on les fait suivre via un throws) ?

3. Enfin, modifier la gestion des exceptions liées à l'accès au fichier pour que l'exception soit accompagnée d'un message d'erreur en français.

4. Remettre individuellement votre rapport en PDF sur Moodle dans l'espace prévu à cet effet. La remise devra s'effectuer le plus tôt possible : si possible pendant la séance, sinon au plus tard le 10 avril à 8h.

D. Pour finir

Lors de la séance précédente, nous avions vu qu'il était important de fermer systématiquement les flux (par exemple les fichiers en écriture). Pour autant, la génération d'une exception dans un bloc try / catch ne permet pas d'exécuter l'intégralité du code prévu par le programmeur. Pour effectuer des opérations qu'il y ait eu levée ou non d'exception, il est possible de compléter le bloc try / catch par l'instruction finally.

Modifier le programme pour que la gestion des exceptions sous forme de blocs try/catch intègrent lorsque nécessaire l'instruction finally.

E. Mise en œuvre : à finir en séance (pour les plus rapides) ou chez soi (pour les autres)

Remarque : cet exercice est indépendant des questions précédentes. Lisez bien tout l'énoncé pour adopter la démarche adéquate dans le processus de développement et test de l'application à réaliser.

On désire développer une petite application qui simule le fonctionnement d'un parking de voitures. On décidera du nombre de places disponibles sur ce parking au départ (il s'agira d'une constante Java private static final int NB_PLACES = …).

On modélisera deux actions possibles :

- Garer une voiture sur le parking : une voiture peut se garer à une place à condition que cette place soit libre (null en pratique). Si la place n'est pas libre, une exception ExceptionParking est lancée avec le message d'erreur "Place déjà occupée".
- Sortir une voiture du parking : consiste à donner un numéro de place de parking et à renvoyer la voiture stationnée à cette place. S'il n'y a pas de voiture garée à la place de parking, une exception ExceptionParking est lancée avec le message d'erreur "Pas de voiture à cette place".

Dans tous les cas, il faudra vérifier que le numéro de place est compatible avec la plage des valeurs possibles (c'est-à-dire 0 <= numéro de place < NB_PLACES). Si le numéro de place n'est pas valable, une exception ExceptionParking est lancée avec le message d'erreur "Numéro de place non valide".

1. Coder une classe Voiture qui contiendra : 3 attributs qui caractérisent une voiture : String marque (ex. Renault), String modèle (ex. Scénic), int puissance (ex. 110) ; un constructeur qui prend 3 paramètres ; une méthode retournant une représentation des attributs d'une voiture : public String toString().
   Tester cette classe en 2-3 lignes.

2. Coder une classe ExceptionParking (qui hérite de Throwable) qui contient simplement un constructeur prenant en paramètre le message d'erreur : public ExceptionParking (String messageErreur). L'unique ligne de code de ce constructeur est l'appel au constructeur de la super-classe (super(messageErreur)).
   Pourquoi procède-t-on de la sorte ?

3. Coder la classe Parking (respecter l'ordre indiqué pour le développement).
   - Déclarer la constante NB_PLACES.
   - Déclarer 1 attribut lesPlaces qui est une référence sur un tableau de Voiture.
   - Ecrire le constructeur qui construit simplement le tableau de Voiture à la taille NB_PLACES. A l'issue de cette construction, le parking est créé et toutes les places sont libres. En effet, on considère qu'une place est libre lorsque la case du tableau correspondant au numéro de place contient la valeur null. Les places de parking sont numérotées de zéro à NB_PLACES-1.
   - Ecrire la méthode public String toString() qui renvoie une chaîne de caractères qui contient l'état du parking. C'est-à-dire pour chaque place de parking, le numéro ainsi que les caractéristiques de la voiture qui l'occupe (ou simplement « place libre » si aucune voiture n'est garée à cette place).
   - Ecrire une méthode privée void numeroValide (int numPlace) throws ExceptionParking qui vérifie que le numéro de place est valide (c'est-à-dire 0 <= numPlace < NB_PLACES). Dans le cas contraire, lance l'exception avec le message "Numéro de place non valide".
   - Ecrire la méthode publique void garer (Voiture voit, int numPlace) throws ExceptionParking qui gare la voiture passée en paramètre à la place numPlace. Vérifier d'abord que le numéro de place est valide (méthode numeroValide(...)) et vérifier ensuite que la place est disponible sinon lancer l'exception avec le message "Place déjà occupée".
   - Ecrire la méthode publique Voiture sortir (int numPlace) throws ExceptionParking qui renvoie la voiture garée au numéro de place passé en paramètre. Vérifier d'abord que le numéro de place est valide et vérifier ensuite qu'une voiture est bien garée à cette place sinon lancer l'exception avec le message "Pas de voiture à cette place". Une fois la voiture sortie, remettre la case à la valeur null puisqu'elle est à nouveau disponible.

4. Ecrire une classe de test (TestParking) qui met en évidence le bon fonctionnement de la classe Parking et des exceptions (lancement des bonnes exceptions avec le bon message). Prendre un parking de taille réduite (NB_PLACES = 2 par ex.) pour faciliter les tests (un parking de 100 places poserait déjà des problèmes à l'affichage !).
   - DES la fin de l'écriture du constructeur et de la méthode toString() de la classe Parking, IL FAUT DEJA tester l'affichage de votre parking qui DOIT indiquer que toutes les places sont libres.
   - Coder la méthode privée numeroValide(...). Celle-ci entant privée, son test se fera à travers les méthodes garer(...) et sortir(...).
   - Coder la méthode garer(...). Considérer 3 cas de test : test du fonctionnement normal sans aucune exception, test avec capture (try/catch) obligatoire de l'exception ExceptionParking avec le message "Numéro de place non valide" (lancement de l'exception dans la méthode numeroValide(...)) et test avec capture obligatoire de l'exception ExceptionParking avec le message "Place déjà occupée".
   - Coder la méthode sortir(...). Considérer 3 cas de test : test du fonctionnement normal sans aucune exception, test avec capture obligatoire de l'exception ExceptionParking avec le message "Numéro de place non valide" et test avec capture obligatoire de l'exception ExceptionParking avec le message "Pas de voiture à cette place".

5. Remettre individuellement l'ensemble de votre application (sources des classes Voiture, ExceptionParking, Parking, TestParking) sous forme d'archive ZIP avec le nom des auteurs dans le dépôt prévu à cet effet sur Moodle, avant lundi 24 avril 2017 à 8h.