Ing39 : Programmation Java

Tp0 : Eclipse et rappels de Java sans objets

V. Aponte

1er septembre 2023

Important : Au moment de réaliser ce Tp vous n'aurez probablement pas encore votre compte personnel. Vous travaillerez alors sur un compte commun à tous les élèves, et il faudra donner des noms différents à vos projets, ou les placer dans un répertoire identifié par votre nom. Créez donc soit un répértroie à votre nom pour tout vos projets, ou accolez votre nom à celui du projet. Exemple : `tp0Aponte`*. Rappel : toutes les ressources pour le cours et les tps sont à trouver sur le site* `deptinfo.cnam.fr`*. Faire une recherche avec le mot clé ING39 puis sélectionner Algorithmique-Programmation. Pour ce Tp vous aurez besoin de comprendre les tableaux et leur répresentation en mémoire (pointeurs). Dans la partie I, on crée un nouveau projet. Dans la partie II, on importe un projet existant.*

Partie I : Création d'un projet Eclipse (en passant par le système de fichiers)

On va créer un projet Maven et apprendre à y ajouter des fichiers pré-existants en passant par le système de fichiers plutôt que par l'IDE. Cela est souvent préférable au copier-coller, qui réalise des réorganisations et des renommages (*refactoring*) qui correspondent rarement à ce que l'on veut faire.

1. Allez dans Eclipse et créez votre premier projet `New` ⇒ `Project` ⇒ `Maven Project`.
2. Cochez `simple project (skip archetype selection)`.
3. Choisissez un répertoire. Attention ce répertoire sera directement la racine du projet, choisissez donc un répertoire VIDE au départ. Au besoin créez le répertoire depuis Eclipse avant de le désigner. Vous devez donner un nom d'artifact `tp0` (accolé de votre nom si necéssaire) et de groupe ou « id » : (`ing39.nom.prenom`) validez.
4. Créez un premier paquetage (package) dans `src/main/java` nommé `bonjour` (en minuscules). Dedans, créez une première classe qui affiche bonjour. Exécutez la.
5. Créez un deuxième paquetage dans `src/main/java` nommé `premcase`. Pour cela, **placez vous dans ce répertoire**. Autrement vous obtiendriez un sous-paquetage, et ce n'est pas l'objectif ici.
6. Allez sur la vue *Navigator*. Elle vous montre le répertoire racine de votre projet et tous ses sous-répertoires.
7. Ouvrez l'explorateur de fichiers de votre système (*dolphin* sous KDE, Linux) et cherchez le répertoire racine de votre projet. Identifiez l'arborescence du répertoire des sources de votre projet qui doit contenir : `tp0/src`.

8. Recupérez le fichier `PremiereCase.java` sur le site du cours. Attention, ce fichier doit avoir exactement le même nom que la classe qu'il contient, y compris mêmes majuscules et minuscules. Si ce n'est pas le cas, RENOMMEZ LE FICHIER (et non pas la classe). Mainteant, en passant par le système de fichiers (*dolphin*) , placez ce fichier dans le sous-répertoire correspondant au paquetage `premcase`. Le chemin du répertoire où placer ce fichier doit se terminer par `src/main/java/premcase` Allez sur Eclipse et vérifiez qu'il est bien visible. Au besoin : `File` ⇒ `Refresh`.
9. Le fichier `PremiereCase.java` contient une erreur (le nom du paquetage ne correspond pas). Corrigez.
10. Exécutez ce fichier.

Partie II : importer un nouveau projet

Importez dans Eclipse le projet `TpBases` se trouvant sur la page du cours (décompressez-le d'abord). Utilisez *File/Import/Existing Maven project* désigné le répertoire. Vous y trouverez plusieurs paquetages nommés `exoN_XXX` avec le code nécessaire à chaque exercice de ce Tp.

Remarque : N'UTILISEZ PAS *File/Open Project From File system*. Eclipse ne détectera pas la configuration maven et risque de ne pas avoir la bonne configuration pour compiler.

Exercice 1 : fonctions statiques sur les caractères

Considérez le programme du paquetage `exo1_static_methods` et répondez aux questions se trouvant dans son code.

Question 1

Expliquez le comportement des méthodes `estMajuscule`, `estMinuscule` et `estLettre`. Ajoutez un commentaire dans l'entête de chaque méthode permettant de décrire sa fonction.

Question 2

Le morceau de code `('A' <= c && c <= 'Z')` est-il une instruction ou une expression ? Quelle différence entre les deux ?

Question 3

La variable c est de type char mais est initialisée à une valeur de type int, expliquez cette particularité.

Question 4

Expliquez le comportement du bout de code ligne 36 : `(int) c` : quelle est sa fonction ici et pourquoi est-il nécessaire ?

Question 5

Expliquez le comportement du main puis exécutez le programme afin de vérifier la justesse de votre réponse.

Question 6

Completez le main de manière à afficher pour chaque caractère s'il est ou non un chiffre. Auparavant vous aurez définit une nouvelle méthode statique `estChiffre` pour tester si un caractère est un chiffre.

Question 7

Allez voir la documentation (oracle) de la classe `Character` de la bibliothèque standard. Remplacez les appels aux méthodes statiques `estMajuscule`, `estMinuscule`, `estChiffre` et `estLettre` par des appels aux méthodes appropriés de la classe `Character`.

Exercice 2 : méthodes non statiques sur les chaînes de caractères

Dans cet exercice vous utiliserez le code du paquetage `exo2_string`. Le type `String` de chaînes de caractères est prédéfini en Java. Il s'agit d'un `type objet` : cela signifie en particulier que la classe `String` contient des méthodes **non statiques**, qui s'appliquent sur les objets de cette classe par une synstaxe particulière. C'est le cas des méthodes `length`, `charAt`, etc : si s est un objet `String` on peut lui appliquer les méthodes `s.length()` pour obtenir la taille de s, et `s.charAt(i)` pour obtenir le caractère se trouvant à la position i de la chaîne s. Comme pour les tableaux, les positions des caractères dans les chaîne débutent à 0.

Question 1

Ecrivez les fonctions statique `estMot`, `contientChar` et `nombreOccChar` dont les contrats sont spécifiés dans le code fourni. **Leur comportement doit correspondre à ce que leur contrat spécifie.**

Question 2

Ajoutez dans le main les appels à ces méthodes qui vous semblent manquer afin de tester tous les cas différents de comportement de ces méthodes.

Question 3

Si cela est possible, remplacez les boucles dans le code de ces méthodes par des appels à méthodes non statiques existantes sur les chaînes (String). Le but ici est de simplifier votre code en vous reposant sur des méthodes déjà fournies dans la bibliothèque standard. Quelles sont les méthodes que vous ne réusissez pas à simplfier ? Attention : la méthode `estMot` se fera toujours avec une boucle, mais qui invoque des méthodes de la bibliothèque standard plutôt que celles fournies ici.

Question 4

Après avoir simplifié vos méthodes, re-exécutez la méthode main. Tous vos tests continuent à se passer correctement ? Si ce n'est pas le cas, cela signifie probablement que vous avez changé le comportement de vos méthodes, qui ne respectent plus la spécification donnée dans leur commentaire. Vous avez donc introduit des bugs !

Exercice 3 : tableaux et références

Les sources de cet exercice se trouvent dans le projet `TpBasesJava`, paquetage `exo3`. Vous y trouverez une classe par exercices. Vous aurez besoin du debuggeur Eclipse et de PythonTutor. Chaque programme est à exécuter en premier dans Eclipse : vous répondrez aux questions posées sous forme de commentaires, et ensuite dans PythonTutor. Cela vous permettra de vérifier la justesse de vos réponses. Certaines lignes de code sont érronées : il faudra donc veiller à tester votre code pas à pas en retirant les lignes éronnées ou en les corrigeant avant de continuer.

Mise en train : petit QCM

Considérez le code suivant :

```
1  int [] xx = {1,7,3};
2  int [] yy;
3  int [] zz = new int [4];
4  double [] w = new double[0];
5  int [] pp = {10};
```

Commencez par faire un dessin de la mémoire du main où le contenu de ces variables est representé. Utilisez ensuite le debugeur de Eclipse (`Run` ⇒ Debug, après avoir posé un premier breakpoint) ou PythonTutor pour vérifier que votre dessin est juste. Pour les énoncés suivants, indiquez s'ils sont vrais (V) ou faux (F) :

1. Le contenu de la variable xx est 1,7, 3
2. Le contenu de la variable xx est une adresse mémoire à partir de laquelle sont stockées ses composantes 1,7, 3.
3. Les composantes de xx ont été crées et initialisées en mémoire avec les valeurs 1, 7 et 3 respectivement.
4. Le contenu de la variable yy est l'adresse mémoire `null`.
5. Les composantes de zz ont été crées mais pas encore initialisées en mémoire.
6. Les composantes de yy ont été crées en mémoire mais ne sont pas encore intialisés.
7. Le tableau zz possède 4 composantes.
8. Le tableau yy ne possède aucune composante et toute tentative d'accès produit une erreur à l'exécution.
9. Le tableau m possède 0 composantes : tout ce qu'on peut faire avec est d'obtenir sa longueur.
10. `System.out.println(xx.length)` affiche la taille du tableau `xx`. Cette taille est 3.
11. `System.out.println(yy.length)` affiche la taille du tableau `xx`. Cette taille est 0.
12. `System.out.println(yy.length)` produit une erreur à l'exécution, car `yy` ne possède aucune composante.

13. `System.out.println(m.length)` affiche la taille du tableau `m`. Cette taille est 0.
14. L'affectation `yy[0]= 20;` est correcte.
15. L'affectation `xx[3]= 20;` est correcte.
16. L'affectation `zz[3]= 20;` est correcte.
17. L'affectation `zz[3]= yy[0]+ 20;` est correcte.
18. L'affectation `zz[3]= zz[0]+ 20;` est correcte.
19. L'affectation entre tableaux n'est possible que s'ils sont de même type et ont la même taille.
20. L'affectation `pp= xx;` est correcte.
21. L'affectation `xx= pp;` est correcte.
22. L'affectation entre tableaux est toujours possible s'ils sont de même type. Le résultat est la recopie de l'adresse contenue dans celui à droite vers le contenu de celui à gauche.
23. Un tableau à deux dimensions doit necéssairement avoir le même nombre de colonnes à toutes les lignes.
24. Un tableau à deux dimensions est en réalité un tableau de tableaux.
25. Un tableau à deux dimensions est en réalité un tableau dont les composantes sont des adresses.
26. Les deux dernières affirmations sont équivalentes.
27. On peut remplacer une ligne d'un tableau à deux dimensions par une autre ligne de taille différente.

Question 1

Cet exercice traite de la declaration, initialisation et parcours de tableaux à une dimension. Considérez la classe `Exo3_question1`.

Sur un papier, commencez par réaliser un dessin avec le contenu des variables tableaux en mémoire après leur déclaration. Vous ferez évoluer votre dessin pour montrer l'état de la mémoire en fin de programme (en ignorant les lignes érronées). Si vous ne savez pas comment faire le dession utilisez PythoTutor qui fera le dessin pour vous (voir à la fin de cette question). Ensuite, à côte de chaque ligne :

- Dans une ligne sans erreur :
  - si la variable affectée est un tableau, explicitez son contenu (pour toutes ses cases) ou indiquez `null` le cas échéant.
  - si on affecte uniquement une case, donnez uniquement la valeur de la case affectée.
- Si la ligne contient une erreur, expliquez sa nature (compilation, exécution) et la raison (typage, conversion, etc). Ex : ERREUR de compilation : variable non déclarée.
- Si la ligne contient plusieurs erreurs, ne signalez qu'une d'entre elles.

Utilisez maintenant PythonTutor pour tester la justesse de vos réponses. Attention : commencez par le début du main dont vous êtes certain qu'il est correct, puis éditez le en ajoutant, une à la fois, la ligne de code que vous voulez tester et en l'enlevant une fois testée, si elle est incorrecte. Sinon, votre programme pourrait ne pas compiler ou échouer.

Question 2

Considérez la classe `Exo3_question2`. Exécutez une première fois sous Eclipse, puis avec PythonTutor. Expliquez les affichages et les erreurs.

Question 3

Considérez la classe `Exo3_question3`. Exécutez une première fois sous Eclipse, puis avec Python-Tutor. Expliquez les affichages et les erreurs.

Exercice 4 : exercices de programmation sur tableux

Le code fourni pour cet exercice contient des squelettes de méthodes (non implantées) dont le fonctionnement est décrit par des contrats en commentaires. Certains de ces contrats pourront être trop imprécis. Dans cet exercice vous allez vous efforcer de suivre soigneusment l'ordre des questions. En clair, il s'agit de bien préciser les contrats, puis de concevoir tous les tests, et seulement APRES, d'implanter les méthodes demandées.

Question 1

Lisez attentivement les contrats, si cela vous semble nécessaire, completez, précisez les contrats. Par exemple si le résultat ou comportement dans certains cas n'est pas suffisamment spécifié.

Question 2

Ajoutez dans le main tous les tests que vous semblerent couvrir les différents de comportement des méthodes. Vous pouvez revenir sur vos réponses dans la question 1 si quelque chose dans le contrat ne vous semble pas assez bien spécifié.

Question 3

Après avoir réalisé 1 et 2 avec application, implantez les méthodes demandées.