Univ. PARIS DESCARTES Programmation avancée et application Licence 3 Informatique Bases de la POO : premières classes 2019

**Exercice I (Utilisation d’une API tierce)**

Les classes up.mi.jgm.maths.Rationnel et up.mi.jgm.maths.Complexe sont disponibles sur Moo dle, dans l’archive Jar nommée maths.jar. La documentation de (la partie publique de) ces classes est donnée ci-dessous.

**public c lass** Rationnel {

*/ \* \**

*\* Cree un Rationne l de numerateur e t denominateur donnes*

*\**

*\* @param num l e numerateur*

*\* @param den l e denominateur*

*\*/*

**public** Rationnel (**long** num, **long** den ) { . . . }

*/ \* \**

*\* Cree un Rationne l ega l a un en t i e r donne*

*\**

*\* @param num l e numerateur*

*\*/*

**public** Rationnel (**long** num) { . . . }

*/ \* \**

*\* Cree un Rationnel , somme de ce Rationne l e t d ’un Rationne l donne \**

*\* @param r l e ra t ionne l qu ’ on a jou te a l ’ o b je t courant*

*\* @return un nouveal ob je t repre sen tan t la somme de ce ra t ionne l par r \*/*

**public** Rationnel addition ( Rationnel r ) { . . . }

*/ \* \**

*\* Cree un Rationnel , produit de ce Rationne l e t d ’un Rationne l donne \**

*\* @param r l e ra t inne l par leque l on mu l t ip l ie l ’ o b je t courant \* @return un nouveal ob je t repre sen tan t l e produit de ce ra t ionne l par r \*/*

**public** Rationnel mul tiplica tion ( Rationnel r ) { . . . }

@Override

**public** S t rin g toS t rin g ( ) { . . . }

}

**public c lass** Complexe {

*/ \* \**

*\* Cons tru i t un nombre complexe a pa r t ie de s e s deux p a r t i e s \**

*\* @param p a r t i eR e e l l e la pa r t ie r e e l l e du nombre*

*\* @param par t ie Imag ina ire la pa r t ie imaginaire du nombre*

Made with LATEX 2*ε* 1/7

*\*/*

**public** Complexe (**double** p a r ti eR e ell e , **double** par tie Imaginaire ) { . . . }

*/ \* \**

*\* Permet d ’ ob ten ir la pa r t ie r e e l l e du nombre*

*\**

*\* @return la pa r t ie r e e l l e du nombre*

*\*/*

**public double** g e tP a r ti eR e ell e ( ) { . . . }

*/ \* \**

*\* Permet d ’ ob ten ir la pa r t ie imaginaire du nombre*

*\**

*\* @return la pa r t ie imaginaire du nombre*

*\*/*

**public double** ge tPar tie Imaginaire ( ) { . . . }

@Override

**public** S t rin g toS t rin g ( ) { . . . }

}

1. Dans une classe utilitaire, définissez une méthode statique qui calcule la puissance *n*-ème d’un rationnel.

2. Dans la même classe utilitaire, définissez :

(a) une méthode statique qui fait la somme de deux nombres complexes. 1 (b) une méthode statique qui fait le produit de deux complexes. 2

**Exercice II (Géométrie et POO)**

On souhaite manipuler des objets du plan.

1. (a) Créez une classe Point qui modélise un point du plan, c’est-à-dire une entité composée d’une abscisse et d’une ordonnée.

(b) Créez une méthode de la classe Point qui prend en entrée un autre Point, et retourne la distance entre les deux points. 3

2. (a) Dans un plan, on peut représenter un disque avec deux informations : les coordonnées de son centre d’une part, et son rayon d’autre part. Créez la classe Disque correspondante.

(b) Un point du plan appartient à un disque si la distance entre ce point et le centre du disque est inférieure ou égale au rayon du disque. Définissez une méthode qui permet de le vé rifier.

(c) Définissez une méthode de la classe Disque qui vérifie si deux disques sont en intersec tion.

3. (a) On peut représenter un vecteur *−−→AB* par les deux points *A* et *B* qui le composent. Définis sez une classe vecteur pour cela.

1. La somme de deux nombres complexes *a +i ×b* et *c +i ×d* est (*a +c*)*+i ×*(*b +d*).

2. Le produit de deux nombres complexes *a +i ×b* et *c +i ×d* est (*a ×c −b ×d*)*+i ×*(*a ×d +b ×c*). 3. La distance entre deux points (*xa*,*ya*) et (*xb*,*yb*) est p(*xb − xa*)~~2~~ *+*(*yb − ya*)~~2~~.

Made with LATEX 2*ε* 2/7

(b) Deux vecteurs sont égaux si la différence entre les coordonnées de leurs extrémités sont égales. Définissez une méthode equals() qui vérifie cela. 4

(c) Une translation consiste à déplacer un point dans la direction et la distance représentées par un vecteur. Ajoutez à la classe Point une méthode qui prend en entrée un vecteur, et qui retourne un nouveau Point correspondant à la translation de l’objet initial par ce vecteur.

(d) La translation d’un disque par un vecteur consiste en un nouveau disque dont on a dé placé le centre, mais en conservant son rayon. Définissez une méthode dans la classe Disque qui effectue la translation du disque par le vecteur donné en paramètre.

**Exercice III (Manipulation de listes et généricité)**

1. Définissez une méthode qui retourne l’élément maximal d’une liste d’Integer. 2. Définissez une méthode qui calcule la moyenne des éléments d’une liste d’Integer. 3. Définissez une méthode qui trie une liste d’Integer.5

4. L’interface java.lang.Comparable<T> définit une unique méthode **public int** compareTo(T o) qui sert à comparer deux objets. Si celui qui appelle la méthode est considéré comme plus petit que o, alors la méthode retourne un nombre négatif, s’il est plus grand un nombre positif. Enfin, la méthode retourne 0 dans le cas où les deux objets sont considérés comme identiques. C’est ce qui permet, par exemple, de comparer des String selon l’ordre lexicograhpique.

(a) Définissez une méthode qui retourne l’élément maximal d’une liste d’objets grâce à l’in terface Comparable.

(b) Définissez une méthode qui trie une liste d’objets grâce à l’interface Comparable.

**Exercice IV (Promotions d’étudiants)**

On souhaite développer un programme qui permet de gérer une promotion d’étudiants. Il doit être possible d’ajouter des étudiants dans la promotion, ainsi que les unités d’enseignement suivies. 6 Pour chaque unité d’enseignement, chaque étudiant a deux notes : un contrôle continu (CC), et un examen (EX). La note finale d’un étudiant dans une unité d’enseignement est max(*E X*,*CC+E X*

2).

Écrivez un programme qui propose à l’utilisateur les fonctionnalités suivantes via une interface textuelle (entrée au clavier et affichage sur la console) :

• Ajout d’un étudiant dans la promotion;

• Ajout d’une note (de contrôle continu ou d’examen) pour un étudiant particulier;

• Afficher l’ensemble des étudiants de la promotion, avec leur moyenne par unité d’enseigne ment;

• Calcul de la moyenne globale de la promotion;

• Calcul de la moyenne par unité d’enseignement;

• Recherche du major de promotion (pour simplifier, nous considérons qu’il n’y a qu’un ma jor, en cas d’ex-aequo un seul est affiché);

• Recherche du major par unité d’enseignement (même remarque sur les éventuels ex-aequo). 4. Formellement, *~~−−→~~AB =~~−−→~~CD* si et seulement si *xB − xA = xD − xC* d’une part, et *yB − yA = yD − yC* d’autre part. 5. Ne pas utiliser de méthode de tri fournie par l’API standard de Java.

6. Pour simplifier, on suppose que tous les étudiants de la promotion suivent exactement les mêmes unités, et que les unités ont le même coefficient.

Made with LATEX 2*ε* 3/7

**Exercice V (Répertoires)**

On veut implémenter une classe RepertoireSimple qui permet de créer des répertoires télépho niques simples, permettant :

• l’enregistrement d’une personne identifiée par son prénom, son nom et son numéro de té léphone,

• et la recherche d’un numéro de téléphone en connaissant l’identité de la personne. La classe RepertoireSimple est utilisée dans la classe suivante :

**public c lass** Tes tReper toire {

**public s t a t i c void** main ( S t rin g [ ] args ) {

RepertoireSimple rep = **new** RepertoireSimple ( ) ;

rep . addPersonne ( "John" , "Lennon" , "0123456789" ) ;

rep . addPersonne ( "Paul " , "McCartney" , "0234567891" ) ;

rep . addPersonne ( "George" , "Harrison " , "0345678912" ) ;

rep . addPersonne ( "Ringo" , " S t a r r " , "0456789123" ) ;

System . out . p rin tln ( rep . chercheNumero ( "John" , "Lennon" ) ) ;

System . out . p rin tln ( rep . chercheNumero ( "Paul " , "McCartney" ) ) ; System . out . p rin tln ( rep . chercheNumero ( " Freddie " , "Mercury" ) ) ; }

}

L’exécution de ce programme donne le résultat suivant :



1. Déduire du code précédent les signatures des méthodes de la classe RepertoireSimple. 2. Implémenter cette classe.

**Exercice VI (HashMap et T9)**

Un peu d’histoire (ou de préhistoire) de la téléphonie mobile : le T9 (*Text on 9 keys*) utilisé pour taper des mots sur un clavier de téléphone repose sur l’association de chaque chiffre du clavier à plusieurs lettres (2 correspond à a, b et c, 3 correspond à d, e et f,... ). Lorsque l’on tape sur une suite de touche, la série de chiffres peut être associée à un mot correspondant.

Avec le clavier ci-contre, une pression sur les touches 2665687 peut 

être associée au mot "bonjour" (le 2 est transformé en b, le pre

mier 6 en o, le deuxième 6 en n,... ). Cependant, certaines suites de

chiffres peuvent correspondre à plusieurs mots : 26663 correspond à

"bonne" (2 *→* b, 6 *→* o, 6 *→* n, 6 *→* n, 3 *→* e) ou à "comme" (2 *→* c,

6 *→* o, 6 *→* m, 6 *→* m, 3 *→* e). Il faut donc mémoriser les différentes

possibilités, et les proposer à l’utilisateur lorsque la situation se pré

sente.

Made with LATEX 2*ε* 4/7

On fournit une méthode statique qui renvoie le chiffre T9 associé à un caractère :

**public s t a t i c byte** ge tChi f freT9 ( **char** c ) {

**switch** ( Character . toLowerCase ( c ) ) {

**case** ’ a ’ : **case** ’b ’ : **case** ’ c ’ :

**return** 2; **case** ’d ’ : **case** ’ e ’ : **case** ’ f ’ :

**return** 3; **case** ’ g ’ : **case** ’h ’ : **case** ’ i ’ :

**return** 4; **case** ’ j ’ : **case** ’ k ’ : **case** ’ l ’ :

**return** 5; **case** ’m’ : **case** ’n ’ : **case** ’o ’ : **return** 6;

**case** ’p ’ :

**case** ’q ’ :

**case** ’ r ’ :

**case** ’ s ’ :

**return** 7;

**case** ’ t ’ :

**case** ’u ’ :

**case** ’ v ’ :

**return** 8;

**case** ’w’ :

**case** ’ x ’ :

**case** ’ y ’ :

**case** ’ z ’ :

**return** 9;

**de fau lt** :

**return** 0;

}

}

1. Définissez une méthode statique qui prend en entrée un mot, et qui retourne la chaîne de caractères T9 correspondante (par exemple, le mot "bonjour" est codé en T9 avec "2665687").

On souhaite maintenant créer un dictionnaire de chaînes T9 grâce à une HashMap<String, ArrayList<String>>, dont les clés sont des chaînes codées en T9, et les valeurs sont les listes de mots correspondant à ces clés. Par exemple, à la clé "26663" correspond la liste qui contient les mots "bonne" et "comme".

2. Définissez une méthode statique qui prend en entrée une HashMap<String, ArrayList<String>> et un String et qui place le String dans la table de hachage :

**public s t a t i c void** en r e gi s t r e r (HashMap<S tring , A r rayLis t <S tring >> dico , S t rin g chaine ) {

*/ / . . .*

}

3. Définissez une méthode qui permet d’obtenir la liste des chaînes de caractères correspondant à une chaîne codée en T9 :

**public s t a t i c** A r r a yLi s t recuperer (HashMap<S tring , A r rayLis t <S tring >> dico , S t rin g chaineT9 ) {

*/ / . . .*

}

Vous pouvez tester les méthodes précédentes avec ce programme :

**public c lass** TestDicoT9 {

**public s t a t i c void** main ( S t rin g [ ] args ) {

Made with LATEX 2*ε* 5/7

HashMap<S tring , A r rayLis t <S tring >> dico = **new** HashMap<S tring , A r rayLis t <S tring > > ( );

en r e gi s t r e r ( dico , "bonjour" ) ;

en r e gi s t r e r ( dico , "bonne" ) ;

en r e gi s t r e r ( dico , "comme" ) ;

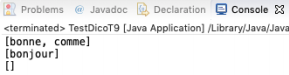
System . out . p rin tln ( recuperer ( dico , "26663" ) ) ;

System . out . p rin tln ( recuperer ( dico , "2665687" ) ) ;

System . out . p rin tln ( recuperer ( dico , "123456" ) ) ;

}

}



**Exercice VII (Jeu de cartes)**

On a vu dans le cours comment utiliser des Enum pour représenter les couleurs dans un jeu de cartes.

1. Définissez une classe Carte qui permet de représenter une carte à jouer. 7

2. Les cartes qu’un joueur possède sont appelées sa main. Créez une classe qui permet de repré senter la main d’un jouer, d’y ajouter une carte ou d’en retirer une carte.

3. On souhaite que l’un joueur puisse trier sa main en utilisant la méthode de tri générique de l’exercice III. Modifiez la classe Carte pour vous en assurer. 8

**Exercice VIII (Jeu de cartes... amélioré)**

On souhaite améliorer la modélisation du jeu de cartes proposée dans l’exercice précédent. Il faut maintenant pouvoir trier les cartes différemment, selon les règles du jeu auquel on joue.

1. Définissez une classe abstraite CarteAbstraite qui permet de représenter les informations de base d’une carte (couleur et valeur).

2. Une classe fille de CarteAbstraite doit correspondre aux cartes associées au système de classe ment d’un jeu particulier. Par exemple :

(a) La classe CarteBasique doit correspondre à un tri simple des cartes, comme vu à l’exer cice VII.

(b) La classe CarteManille permet de trier les cartes comme à la manille. 9

3. Assurez vous qu’il est possible pour un joueur de trier les cartes de sa main, dans le cas basique et dans le cas de la manille.

7. On utilisera les cartes d’un jeu classique : coeur, carreau, trèfle et pique pour les couleurs, et les nombres de 1 à 10, ainsi que valet, dame et roi pour les valeurs.

8. On utilisera un ordre « naturel » : si deux cartes sont de la même couleur, on les compare suivant leurs valeurs (l’as est le plus petit, puis 2, puis 3,..., puis le roi qui a la plus grande valeur), et si elles ont une couleur différente, le coeur est plus petit, puis carreau, puis trèfle, puis pique.

9. Voir l’ordre des cartes : https://fr.wikipedia.org/wiki/Manille\_(jeu)

Made with LATEX 2*ε* 6/7

**Exercice IX (POO et chimie)**

Un atome est composé d’un noyau, qui se décompose en protons et neutrons, autour duquel tournent des électrons. Le nombre de protons est égal au nombre d’électrons (*Z*), mais le nombre de neutrons (*N*) peut être différent. Les atomes ont un nom, ainsi qu’une abbréviation officielle. Une molécule peut être vue comme un groupement d’atomes. On peut représenter une molécule sous la forme *XnYm* ...*Zl* où *X*,*Y* ,...,*Z* sont les atomes qui la composent, et *n*,*m*,...,*l* sont les quantités de chaque atome.

1. Créez une classe qui permet de représenter un atome, et d’afficher les informations qui le caractérisent.

2. Créez une classe qui permet de représenter une molécule, et d’afficher les informations qui la caractérisent.

3. Définissez une méthode qui permet de déterminer le nombre de protons dans une molécule.

Vos classes doivent être utilisables avec le code suivant :

**public c lass** TestChimie {

**public s t a t i c void** main ( S t rin g [ ] args ) {

Atome hydro = **new** Atome ( 0 , 1 , "Hydrogene" , "H" ) ;

Atome oxy = **new** Atome ( 8 , 8 , "Oxygene" , "O" ) ;

System . out . p rin tln ( hydro ) ;

System . out . p rin tln ( oxy ) ;

Molecule eau = **new** Molecule ( ) ;

eau . addAtome ( hydro , 2 ) ;

eau . addAtome ( oxy ) ;

System . out . p rin tln ( eau ) ;

System . out . p rin tln ( eau . getNbProtons ( ) ) ;

}

}

Made with LATEX 2*ε* 7/7