Contrôle intermédiaire 2 - 2017

Durée 2 h.

Programme : Tout le début du cours jusqu'à la généricité (compris).

Documents autorisés : Supports de cours distribués pendant le cours (mais **les TPs et corrections de TPs ne sont pas autorisés**).

Exercice 1 :

Vous allez écrire des classes `Personne`, `Etudiant` et `Enseignant` dans le paquetage `fac.personne`.

Un étudiant a un nom et des notes. Une note est représentée par un `double`. Vous utiliserez une collection (pas un tableau) pour conserver les notes des étudiants.

Un enseignant a un nom et une adresse (uniquement la ville sous la forme d'une `String`).

`Etudiant` et `Enseignant` héritent de `Personne`.

La méthode `main` suivante doit pouvoir s'exécuter avec les classes que vous allez écrire. N'oubliez pas d'écrire dans vos classes tout ce qui est nécessaire, **en particulier pour la boucle « for-each »**. Dans vos classes, n'ajoutez rien d'autre que ce qui est indispensable à l'exécution de cette méthode `main`.

```
public static void main(String[] args) {
  Etudiant etudiant = new Etudiant("Bob");
  etudiant.ajouterNote(10.5);
  etudiant.ajouterNote(12);
  Enseignant enseignant = new Enseignant("John", "Paris");
  System.out.println("Les notes de " + etudiant.getNom() + " : ");
  for (double note : etudiant) {
    System.out.println(note);
  }
  System.out.println("Adresse de " + enseignant.getNom() + " : "
    + enseignant.getAdresse());
}
```

Est-ce qu'on aurait pu écrire

« `Personne etudiant = new Etudiant("Bob");` » ?

Pourquoi ?

Suite au verso de la feuille

Exercice 2

Ecrivez une classe `Universite` dans le paquetage `fac`. Une université a un nom de type `String`. La classe a un constructeur et les méthodes suivantes :

- `ajouterPersonne` pour ajouter un étudiant ou un enseignant dans l'université. Vous utiliserez une Map pour enregistrer l'étudiant ou l'enseignant dans l'université. Cette Map utilisera le nom comme clé.
  Une exception `IllegalArgumentException` est lancée s'il existe déjà une personne avec le même nom dans l'université. Rappel : `IllegalArgumentException` est une exception qui n'est pas contrôlée par le compilateur. Le message associé à cette exception sera le suivant :
  <<nom université>> a déjà une personne nommée <<le bon nom>>
- `getPersonnes` qui retourne une collection de tous les enseignants et étudiants de l'université.
- `getEtudiants` qui retourne une liste des **étudiants** de l'université (type retour `List<Etudiant>`. Aide : vous aurez besoin de `instanceof` (pour savoir si une personne est un étudiant) et d'un cast.

Où avez-vous utilisé la généricité dans votre code ? Expliquez pourquoi vous avez choisi ces paramètres de type.

Exercice 3

Dans le même paquetage que `Universite`, écrivez une classe `TestUniversite` avec une méthode `main` qui crée une université et ajoute un enseignant et un étudiant dans l'université.

La méthode `main` ajoute un 2ème étudiant qui a le même nom que l'enseignant et elle affiche le message d'erreur associé à l'exception lancée par `ajouterPersonne`.

Elle affiche ensuite le nom de tous les enseignants et étudiants ajoutés dans l'entreprise (un nom par ligne).

Correction

Exercice 1

```
package fac.personne;

/**
 * Une personne.
 */
public class Personne {
  private String nom;

  public Personne(String nom) {
    this.nom = nom;
  }

  public String getNom() {
    return this.nom;
  }

}

package fac.personne;


/**
 * Un enseignant.
 */
public class Enseignant extends Personne {
  private String adresse;

  public Enseignant(String nom, String adresse) {
    super(nom);
    this.adresse = adresse;
  }

  public String getAdresse() {
    return adresse;
  }

}

package fac.personne;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

/**
 * Un étudiant.
 */
public class Etudiant extends Personne implements Iterable<Double> {
  private final List<Double> notes = new ArrayList<>();

  public Etudiant(String nom) {
```

```
    super(nom);
  }

  public void ajouterNote(double note) {
    this.notes.add(note);
  }

  @Override
  public Iterator<Double> iterator() {
    return notes.iterator();
  }

}
```

Est-ce qu'on aurait pu écrire
« `Personne etudiant = new Etudiant("Bob");` » ?
Pourquoi ?

Réponse : Oui car Etudiant est un sous-type de Personne puisque Etudiant hérite de Personne.

Exercice 2

```
package fac;

import fac.personne.Etudiant;
import fac.personne.Personne;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * Une université.
 */
public class Universite {
  private String nom;
  private Map<String, Personne> personnes = new HashMap<>();

  public Universite(String nom) {
    this.nom = nom;
  }

  public String getNom() {
    return nom;
  }

  public void ajouterPersonne(Personne personne) {
    if (personnes.containsKey(personne.getNom())) {
```

```
      throw new IllegalArgumentException(this.nom
      + " a déjà une personne nommée " + personne.getNom());
    }
    personnes.put(personne.getNom(), personne);
  }

  public Collection<Personne> getPersonnes() {
    return personnes.values();
  }

  public List<Etudiant> getEtudiants() {
    List<Etudiant> etudiants = new ArrayList<>();
    for (Personne personne : personnes.values()) {
      if (personne instanceof Etudiant) {
        etudiants.add((Etudiant) personne);
      }
    }
    return etudiants;
  }
}
```

Où avez-vous utilisé la généricité dans votre code ? Expliquez pourquoi vous avez choisi ces paramètres de type.

Réponse : La généricité a été utilisée pour désigner le type des objets contenus dans la collection retournée par getPersonnes(). Cette collection retourne des objets de type Personne (Etudiant ou bien Enseignant).

Elle a aussi été utilisée pour la variable d'instance personnes qui est une Map dont les clés sont de type String (les noms des personnes) et les valeurs de type Personne.

Et elle a aussi été utilisée pour le type retour de getEtudiants() qui est une liste d'Etudiant, pas de Personne ;

Exercice 3

```
package fac;

import fac.personne.Enseignant;
import fac.personne.Etudiant;

public class TestUniversite {
  public static void main(String[] args) {
    Universite ufe = new Universite("UFE");
  try {
    ufe.ajouterPersonne(new Etudiant("Bob"));
    ufe.ajouterPersonne(new Enseignant("Ric", "Nice"));
    ufe.ajouterPersonne(new Etudiant("Ric"));
  } catch (IllegalArgumentException e) {
```

```
      System.err.println(e.getMessage());
    }
    for (Etudiant e : ufe.getEtudiants()) {
      System.out.println(e.getNom());
    }
  }
}
```

Remarque : il est plus logique d'encadrer tous les ajouts de personnes dans l'université par un bloc try – catch (dans un « vrai » programme, on ne sait pas quand l'exception va être lancée, sinon, on s'arrangerait pour qu'elle ne le soit pas) mais il ne sera pas compté faux si seulement le dernier ajout est dans un bloc try – catch.