UNIVERSITÉ CAEN NORMANDIE

UFR des Sciences

Université de Caen Normandie
Département mathématiques-informatique
2018–2019, 1re session
17 décembre 2017
Bruno Zanuttini

L2 informatique, L2 mathématiques

Examen

Unité M.MIM3A1 : Introduction à la programmation orientée objet
2 h — Tous documents autorisés

Chaque candidat doit, au début de l'épreuve, porter son nom dans le coin de la copie qu'il cachera par collage après avoir été pointé. Il devra en outre porter son numéro de place sur chacune des copies, intercalaires, ou pièces annexées.

1 Notions fondamentales (10 points)

Définitions de classes et interfaces Pour la question 1, on considère les définitions d'interfaces et de classes suivantes :

```
public interface I {
  public void f();
}

public class C1 implements I {
}

public abstract class A1 implements I {
}

public class C2 extends A1 {
}

public class C3 extends A1 {
  @Override
  public void f() {}
}

public abstract class A2 implements I {
  @Override
  public void f() {}
}

public class C4 extends A2 {
}

public class C5 extends A2 {
  @Override
  public void f() {}
}
```

Question 1 (4 points) *Lesquelles des huit définitions ci-dessus sont correctes (c'est-à-dire qu'elles sont compilées sans erreur) ? Justifier.*

Types Pour la question 2, on considère les définitions suivantes :

```
public interface I {
  public void f();
}

public abstract class A implements I {
  @Override
  public void f() {}
}

public class C extends A {
  @Override
  public void f() {}
}

public class E {
  public void g(A a) {
    a.f();
  }
}
```

On rappelle qu'en l'absence d'un constructeur explicite, toutes les classes ont un constructeur par défaut, ne prenant pas d'argument et ne faisant rien de particulier.

Question 2 (2,5 points) *Lesquelles des dix suites d'instructions suivantes sont correctes (c'est-à-dire qu'elles sont compilées sans erreur) ? Justifier.*

1. `E e = new E(); I i = new I(); e.g(i);`
2. `E e = new E(); I i = new A(); e.g(i);`
3. `E e = new E(); I i = new C(); e.g(i);`
4. `E e = new E(); A a = new I(); e.g(a);`
5. `E e = new E(); A a = new A(); e.g(a);`
6. `E e = new E(); A a = new C(); e.g(a);`
7. `E e = new E(); C c = new I(); e.g(c);`
8. `E e = new E(); C c = new A(); e.g(c);`
9. `E e = new E(); C c = new C(); e.g(c);`
10. `C c = new C(); E.g(c);`

Héritage Pour la question 3, on considère les définitions de classes suivantes :

```
public class A {
  protected int a;

  public A () {
    this.a = 0;
  }
}

public class B extends A {
  protected int b;

  public B (int b) {
    super();
    this.b = b;
  }
}
```

Question 3 (1,25 point) *Lesquelles de ces affirmations sont correctes ? Justifier.*

1. *Les instances de la classe* `B` *ont un attribut* `a`.
2. *Les instances de la classe* `B` *ont un attribut* `b`.
3. *On peut construire une instance de* `A` *avec l'appel* `new A()`.
4. *On peut construire une instance de* `A` *avec l'appel* `new A(0)`.
5. *On peut construire une instance de* `B` *avec l'appel* `new B(1)`.

Égalité Enfin, pour la question 4, on considère les déclarations suivantes :

```
A a1 = new A(1);
A a2 = new A(1);
A a3 = a1;
List<A> lst = new ArrayList<>();
lst.add(a1);
```

en supposant que la classe `A` est définie par

```
public class A {
  private int attr;

  public A (int attr) {
    this.attr = attr;
  }

  @Override
  public boolean equals (Object other) {
    return
        other != null
        && other instanceof A
        && this.attr == ((A)other).attr;
  }

  [...]
}
```

Question 4 (2,25 points) *Lesquels des appels suivants retournent* `true` *? Justifier.*

1. `a1 == a2`
2. `a1 == a3`
3. `a2 == a3`
4. `a1.equals(a2)`
5. `a1.equals(a3)`
6. `a2.equals(a3)`
7. `lst.contains(a1)`
8. `lst.contains(a2)`
9. `lst.contains(a3)`

2 Conception (10 points)

L'objectif de l'exercice est de concevoir des classes et interfaces pour une application permettant de traduire des phrases simples (sujet, verbe, complément, comme « Les chats chassent les souris ») d'une langue à une autre (appelées « langue d'entrée » et « langue de sortie », respectivement). Pour toutes les questions, on pourra répondre soit en donnant un diagramme UML des classes, soit en donnant du code Java ou Python, soit en donnant du pseudo-code, au choix.

On représentera les langues, les phrases, les groupes de mots et les mots par des chaînes de caractères.

Question 5 (1 point) *Proposer une interface* `Traducteur` *fournissant des méthodes donnant accès à la langue d'entrée et à la langue de sortie du traducteur, ainsi qu'une méthode* `traduire` *permettant de traduire une phrase de la langue d'entrée en une phrase de la langue de sortie.*

Par exemple, si l'on a une instance `i` *d'une classe* `FrançaisVersAnglais` *implémentant l'interface* `Traducteur`*, on veut notamment une méthode telle que l'appel* `i.methode()` *retourne la chaîne de caractères* `"français"`*, et on veut que l'appel* `i.traduire("Les chats mangent les souris")` *retourne la chaîne* `"Cats eat mice"`*.*

Question 6 (3 points) *Proposer une classe implémentant l'interface* `Traducteur`*, et permettant de représenter des traducteurs utilisant une langue « pivot ».*

Par exemple, une instance de cette classe pourrait être un traducteur permettant de traduire du polonais vers le français en utilisant l'anglais comme langue pivot. Il s'agirait donc pour cette instance d'utiliser un traducteur du polonais vers l'anglais et un traducteur de l'anglais vers le français.

Donner les attributs, les méthodes et le constructeur de votre classe, ainsi que la définition des méthodes et du constructeur.

On souhaite maintenant représenter des traducteurs opérant séparément sur le sujet, le verbe et le complément. On note que le traitement du verbe dépend du sujet ; par exemple, si l'on traduit la phrase française « Je mange un gâteau », on ne peut traduire le verbe « mange » que si l'on sait que le sujet est « je » (et non « il »).

Question 7 (1 point) *Proposer une classe* `Verbe` *permettant de représenter un verbe par son infinitif (par exemple « manger » en français ou « to eat » en anglais), la personne à laquelle il est conjugué (l'entier* `1` *pour la première personne du singulier, l'entier* `2` *pour la deuxième, etc.) et le temps auquel il est conjugué (la chaîne de caractères* `"passé"`*,* `"présent"` *ou* `"futur"`*). Donner les attributs et le constructeur.*

Question 8 (3 points) *En utilisant la classe* `Verbe`*, proposer une classe abstraite implémentant l'interface* `Traducteur`*, indépendamment de la langue, et permettant de représenter un traducteur opérant en trois temps :*

1. *découpage de la phrase donnée dans la langue d'entrée en un sujet, un verbe et un complément,*
2. *traduction indépendante des trois parties,*
3. *reconstitution de la phrase dans la langue de sortie, en concaténant les trois parties traduites.*

La traduction de chacune des parties est évidemment dépendante de la langue.

Par exemple, pour la phrase « Les chats mangent les souris », la traduction découperait d'abord la phrase en (1) sujet = « Les chats », (2) verbe = « manger, 6^{e} personne, présent », et (3) complément = « les souris », puis traduirait chaque partie indépendamment, obtenant (1) traduction du sujet = « Cats », (2) traduction du verbe = « eat », et (3) traduction du complément = « mice », et retournerait enfin la concaténation de ces trois traductions, « Cats eat mice ».

Donner les méthodes de votre classe, en spécifiant pour chacune si elle est concrète ou abstraite, et donner la définition des méthodes concrètes.

On suppose maintenant que l'on a accès à un package `translation`, fournissant notamment une classe `ItalianToSpanish` contenant une méthode de signature `public static String translate(String italianSentence)`, qui retourne, pour toute phrase donnée en italien, sa traduction en espagnol.

Question 9 (1 point) *Proposer une classe* `ItalienVersEspagnol` *implémentant l'interface* `Traducteur` *et réalisant la traduction pour la langue d'entrée* `"italien"` *et la langue de sortie* `"espagnol"`*, en utilisant la classe* `ItalianToSpanish` *(sans la modifier). Donner la définition complète de la classe (attributs, constructeur et méthodes éventuels, ainsi que les définitions de ces dernières).*
Indication : *Créer la classe* `ItalienVersEspagnol` *en « adaptant » la classe* `ItalianToSpanish` *à l'interface* `Traducteur`*, comme vu en cours.*

Question 10 (1 point) *Quel comportement proposez-vous pour la méthode* `traduire` *de l'interface* `Traducteur`*, lorsqu'elle reçoit en entrée une chaîne de caractères ne représentant* pas *une phrase de type sujet-verbe-complément de la langue d'entrée ? Par exemple, quel comportement proposez-vous si la méthode de traduction de la classe* `FrançaisVersAnglais` *est appelée avec l'argument* `"A man will look at the painting"` *?*