TP3 Développement Orienté Objet

Table des matières

I. Mise en place des classes....2
1. Vérifiez la structure de votre sous-répertoire class. Qu'observez-vous ? Comment javac gère la notion de package ?....2
2. Exécutez ce code. Faites état de l'affichage obtenu....2
II. Héritage et chaînage des constructeurs....2
1. Détaillez avec vos connaissances et en vous aidant de l'affichage produit à l'exécution comment sont chaînés les constructeurs :....2
2. Que se passe-t-il si l'on met en commentaire l'instruction super(-10); dans le constructeur de la classe C puis que l'on recompile la classe C ? Expliquez le problème....3
3. Que se passe-t-il si cette fois ci, on place cette instruction super(-10) ; du constructeur de la classe C après l'affichage (donc en deuxième instruction). Expliquez le problème....3
III. Héritage et réutilisation de code avec et sans usufruit....3
1. Rajoutez dans la classe C une méthode de signature public void m5() dont l'objectif est de :...3
2. Rajoutez dans la classe B une méthode de signature public void m6()....4
IV. Accès inter-objets avec ou sans accès d'héritage....5
1. Complétez le code du main() pour vérifier quels sont les attributs et les méthodes (héritées ou non) d'une instance de la classe C qui sont appelables depuis le main(). Ce faisant vous étudiez les accès inter-objets qui sont licites entre deux objets instances de deux classes du même package (ici les classes C et E du package client)....5
2. Même chose pour A....5
3. Sur la base des expérimentations menées vérifiez si les 3 tableaux ci-dessous sont corrects.….5
V. Premier pas dans l'héritage et la redéfinition....7
VI. Premier pas dans l'héritage et le sous-typage....7
1. Règle de sous-typage par héritage....7
2. Demandez à chatGPT de vous expliquer simplement ce phénomène (précisez lui bien dans votre prompt que C est une sous-classe Java de A)....7
3. Demandez à chatGPT de vous expliquez simplement ce qui se passe. Que conclure sur l'usage du transtypage (cast) en Java ?....7

I. Mise en place des classes

1. Vérifiez la structure de votre sous-répertoire class. Qu'observez-vous ? Comment javac gère la notion de package ?

Les classes sont stockées dans deux différents dossiers. Javac regarde le package (dossier) où doit être mis le .class et le met dedans ou le créer et le met dedans si le dossier n'existe pas.

2. Exécutez ce code. Faites état de l'affichage obtenu.

```
aryouko@aryouko-IdeaPad-1-14ALC7:~/Documents/GitHub/Prog/R2.01/TPs/TP3/ws$ java
client/E
Constructeur de A
Constructeur de A
Constructeur de B
Constructeur de A
Constructeur de B
Constructeur de C
Constructeur de A
```

II. Héritage et chaînage des constructeurs

1. Détaillez avec vos connaissances et en vous aidant de l'affichage produit à l'exécution comment sont chaînés les constructeurs :

- de B et A : chaînage automatique car le constructeur de la classe mère A n'a pas de paramètre
- de C et B : chaînage explicite car le constructeur de la classe mère B attend un paramètre
- de D et A : chaînage automatique car le constructeur de la classe mère A n'a pas de paramètre
- de E avec Object : chaînage automatique car le constructeur de la classe mère Object n'a pas de paramètre
- De A avec Object : chaînage automatique car le constructeur de la classe mère Object n'a pas de paramètre

2. Que se passe-t-il si l'on met en commentaire l'instruction super(-10); dans le constructeur de la classe C puis que l'on recompile la classe C ? Expliquez le problème.

Le problème est que C est une sous-classe de B et B a un constructeur avec un paramètre, le chaînage ne peut donc pas être fait automatique et doit être fait explicitement. C'est donc pour ça qu'on a une erreur quand on met en paramètre le chaînage explicite.

3. Que se passe-t-il si cette fois ci, on place cette instruction super(-10) ; du constructeur de la classe C après l'affichage (donc en deuxième instruction). Expliquez le problème

L'instruction « super(-10) » doit être fait au début du constructeur.

III. Héritage et réutilisation de code avec et sans usufruit

1. Rajoutez dans la classe C une méthode de signature public void m5() dont l'objectif est de :

▪ vérifier quels sont les accès d'héritage qui sont autorisés dans C (donc avec usufruit) sur tous les membres hérités de A et B. Mettez ensuite en commentaire les accès refusés par le compilateur.
▪ mettre en évidence que même l'attribut a1 est bien présent dans une instance de la classe C et que l'on peut même obtenir sa valeur et l'afficher

```
public void m5(){
    //this.a1 = 2; a1 has private access in A
    //this.a2 = 2; a2 is not public in A; cannot be accessed from outside package
    this.a3 = 2;
    this.a4 = 2;
    //this.b1 = 2; b1 has private access in B

    System.out.println(this.getA1());
    this.m1();
    this.m2();
    //this.m3(); package privacy (accessible que par les classes d'un meme package)
    //this.m4(); private access
}
```

```
getA1 de A
1
m1 de C
m2 de B
m2 de A
```

Accès à a1.

2. Rajoutez dans la classe B une méthode de signature public void m6()

```
public void m6(){
    //this.a1 = 2; a1 has private access in A
    this.a2 = 2;
    this.a3 = 2;
    this.a4 = 2;

    System.out.println(this.getA1());
    this.m3();
    //this.m4();
}
```

IV. Accès inter-objets avec ou sans accès d'héritage

1. Complétez le code du main() pour vérifier quels sont les attributs et les méthodes (héritées ou non) d'une instance de la classe C qui sont appelables depuis le main(). Ce faisant vous étudiez les accès inter-objets qui sont licites entre deux objets instances de deux classes du même package (ici les classes C et E du package client).

```
//unC.a1 = 2; a1 has private access in A
//unC.a2 = 2; a2 is not public in A; cannot be accessed from outside package
//unC.a3 = 2; a3 has protected access in A
unC.a4 = 2;
//unC.b1 = 2; b1 has private access in B

System.out.println(unC.getA1());
unC.m1();
//unC.m2(); m2() has protected access in B
//unC.m3(); package privacy (accessible que par les classes d'un meme package)
//unC.m4(); private access

    unC.m7();
    unC.m8();
```

2. Même chose pour A

```
//unA.m7();
//unA.m8();
```

3. Sur la base des expérimentations menées vérifiez si les 3 tableaux ci-dessous sont corrects.

Accès d'héritage avec usufruit (oui/non) dans le code de B sous classe directe ou indirecte de A

Visibilité Java du membre de A	**A et B sont dans le même package**	**A et B dans des packages ≠**
private	non	Non
(rien)	oui	Non
protected	oui	Oui
public	oui	Oui

Accès inter-objets direct autorisé (oui/non) si un objet B est client d'un objet A d'un membre défini par A

Visibilité Java du membre de A	**A et B sont dans le même package**	**A et B dans des packages ≠**
private	Non (oui si deux instances de la même classe A=B!)	Non
(rien)	oui	Non
protected	oui	Non
public	oui	Oui

Accès inter-objets (indirect) autorisé (oui/non) si un objet B est client d'un objet A d'un membre dont A a hérité d'une classe C (c'est la visibilité de C qui sert de référence !)

Visibilité Java du membre de A	**A et B sont dans le même package**	**A et B dans des packages ≠**
private	non	Non
(rien)	oui	Non
protected	oui	Non
public	oui	Oui

Avec les expériences précédentes, je n'identifie pas d'erreurs sur les tableaux

V. Premier pas dans l'héritage et la redéfinition

A : « m2 de A »
B : « m2 de B »
C : « m2 de A »
D : « m2 de D » met en public en plus de changé l'affichage donc accès

Si une classe ne redéfinit pas une méthode ce sera la méthode de la classe mère qui sera utilisé

VI. Premier pas dans l'héritage et le sous-typage

1. Règle de sous-typage par héritage

unA est un super classe de B, D et C donc unA =unD/C/B est vrai mais l'inverse non car une sous-classe ne peut pas se transformer en super classe de risque de causer de erreur donc unC ne peut-être égale à unA. unA étant la super classe il ne peut accéder à une méthode d'un sous classe (m5 de C).

```
unA=unB;
unA=unD;
unB=unC;
unA=unC;
//unA.m5();
//unC=unA;
```

2. Demandez à chatGPT de vous expliquer simplement ce phénomène (précisez lui bien dans votre prompt que C est une sous-classe Java de A)

L'exécution aurait **pu fonctionner**, mais Java **bloque dès la compilation** pour éviter d'appeler une méthode qui **n'existe pas dans la classe de référence (A)**. Si vous voulez vraiment exécuter `m5()`, vous devez **forcer le typage** avec un cast explicite

3. Demandez à chatGPT de vous expliquez simplement ce qui se passe. Que conclure sur l'usage du transtypage (cast) en Java ?

- **Le cast est nécessaire pour récupérer des méthodes d'une sous-classe à partir d'une référence de type parent.**
- **Un cast invalide peut provoquer une `ClassCastException` à l'exécution.**
- **Toujours vérifier que l'objet réel est bien du bon type avant de caster**
- **Le transtypage ne change pas la nature de l'objet, il ne fait que "forcer" la lecture de son type différemment.**

➡️ **En Java, le transtypage est puissant mais doit être utilisé avec précaution pour éviter des erreurs d'exécution**