**UB2/Faculté Math & Info/Département Informatique Année 2016/2017,**

**Filière/Module : 2ème LMD ACA/P.O.O Date 31/01/2017**

**Enseignant : N.Kadache Durée : 1 h 30 mn**

**Examen Final**

**Questions de Cours (2+4+4+2 =12 Pts):**

Répondez par des phrases brèves et précises :

**Q1)** Quel est l’intérêt de la séparation entre l’interface (description) d’un TAD et de son implémentation ?

**Q2)** Donnez une définition aux termes : **Classe**, **état d’un objet**, **méthode abstraite** et **méthode redéfinie**.

**Q3)** Expliquez par un exemple la procédure de factorisation des classes.

**Q4)** Quels sont les avantages et les inconvénients de l’invocation statique des méthodes (invocation de la méthode du type statique) et l’invocation dynamique (invocation de la méthode du type courant).

**Exercice (4+1.5+2.5=8Pts):**

Ouest

Nord

Sud

3

2

nom :Ri

x :2,

Y :3

Sens :1

Dans un plan 2D, des robots sont définis (voir figure) par :

* Un nom résultant de la concaténation de la lettre R avec un numéro

séquentiel commençant par 1 (exemple R1, R2 ,R3 ,…) selon

l’ordre de création des robots

* Des coordonnées x et y (entiers) de chaque robot.
* Un entier qui représente un sens d’orientation

Est

(0 : Est, 1 : Nord, 2 : Ouest, 3 : Sud).

Un robot est crée avec des coordonnées initiales données(en paramètres),

Un sens d’orientation par défaut à l’est (sens=0), son nom bien entendu

est généré automatiquement, les coordonnées et le sens ne peuvent être modifiés que

par les 2 méthodes :avancer() et tourner() :

* **avancer()**: permet d’avancer de 1 (une seule) position dans le sens d’orientation du robot.
* **tourner()**: permet de faire tourner le robot de 90° à droite (Est ->Nord->Ouest-Sud).

Une troisième méthode **afficherEtat()** : permet d’afficher le nom, les coordonnées et le sens d’orientation d’un robot en une seule ligne.

**Q1)** Selon la description précédente, Donnez le code Java de la classe Robot (on suppose qu’une position peut contenir plusieurs robots) .

**Q2)** En utilisant la méthode avancer() de la question Q1, ajoutez une méthode **avancer(int nbpas)** qui permet à un robot d’avancer de nbpas dans le sens de son orientation.

**Q3)** On suppose maintenant qu’une position ne peut contenir qu’un seul robot à la fois. Proposez une nouvelle implémentation à la classe Robot.

**N.B :**

**- Répondez sur deux copies d’examen différentes, une pour les questions de cours et l’autre pour l’exercice.**

**- Précisez votre groupe dans les 2 copies.**

**Corrigé-type**

**Examen Final-POO-2ème LMD**

**Q1)** l’intérêt de la séparation entre l’interface (description) d’un TAD et de son implémentation :

* + Meilleure (Lisibilité, abstraction, organisation et réutilisation) de la description. ( 1Pt)
	+ Evolution (changement) de l’implémentation du TAD indépendamment de sa description. (1Pt)

**Q2)**

**Classe :** Abstraction d’un ensemble d’objets ayant la même structure d’état et le même comportement (Moule, Modèle, ensemble d’attributs et de méthodes).(1Pt)

**Etat d’un objet :** la valeur de ses attributs à un instant t. (1Pt)

**Méthode abstraite :** Méthode sans implémentation (implémentation =corps, instructions, réalisations). (1Pt)

**Méthode redéfinie :** méthode reprise dans les classes dérivées (héritières) avec le même entête et des implémentations différentes. (1Pt).

**Q3)** Factorisation :

Rassembler les caractéristiques communes (attributs et méthodes) de plusieurs classes dans une classe abstraite qui servira de super-classe pour ces dernières. (1Pt)

Exemple : (3 Pt)

* Avant factorisation :

class B {

Attributs : att1,att2,att3

Méthodes : m1,m2

}

class C {

Attributs : att1,att2,att4 // att1 et att2 sont du même type que dans B

Méthodes : m1,m3 // m1 possède la même signature que dans B

 // et une implémentation éventuellement différente

}

* Après Factorisation :

Class A {

Attributs : att1,att2

Méthode abstraite m1 //

}

class B extends A {

Attributs : att3

Méthodes : m1,m2 // m1 redéfinie avec implémentation de B

}

class C extends A {

Attributs : att4

méthodes : m1,m3 // m1 redéfinie avec implémentation de C

}

**Q4)**

Invocation dynamique :

+ Evolution(changement du service) sans changement de l’utilisateur.(0.5pt)

- Lenteur (temps d’exécution lent) (0.5pt)

Invocation statique :

+ Efficace (temps d’exécution rapide). (0.5pt)

- Couplage entre service et utilisation (changement de l’un 🡺 changement de l’autre). (0.5pt)

**Exercice :**

**Q1)**

public class Robot {

private static int seq=1; // 0.25 Pt sur l’attribut statique de la séquence

private String nom; // 0.25 Pt sur la déclaration des attributs

private int x;private int y;

private int sens;

public Robot(int x,int y){ // 1Pt

 this.nom="R"+(seq++);

 this.x=x;

 this.y=y;

 this.sens=0;

}

public void avancer(){ // 1Pt ,

 switch(sens){ // ou avec des instructions if

 case 0: {x++;break;}

 case 1: {y++;br eak;}

 case 2: {x--;break;}

 case 3: {y--; break;}

 }

}

public void tourner(){// 1Pt

 sens=(sens+1)%4; // ou avec les instructions if ..

}

public void afficherEtat(){// 0.5 Pt

 System.out.println(nom+":"+x+","+y+"Sens:"+sens);

}

}

**Q2)**

public void avancer(int nbpas){ // 2Pts

for(int i=1 ;i<=nbpas ;i++) // ou for (int i=0 ;i<nbpas ;i++) i.e : nombre itérations de avancer() = nbpas

 this.avacer() ;

}

**Q3) // 2 Pts, 1 Pt sur une idée correcte, 1 Pt sur le code**

Plusieurs solutions sont envisageables :

Solution 1 : dans la classe Robot on ajoute :

* Un tableau de grande capacité qui va contenir les différents objets créés
* Les accesseurs getX() et getY dans la classe Robot
* La méthode avancer() : on doit s’assurer que la futur position est vide
* Le constructeur : test si les coordonnées initiales ne contient pas un robot !!!

public class Robot {

 private static Robot[] robots=new Robot[1000]; // tableau contenant tous les objets robots

 private static int seq=1;

 private String nom;

 private int x;

 private int y;

 private int sens;

 public Robot(int x,int y){

 robots[seq-1]=this; // mettre l'objet robot dans le tableau

 this.nom="R"+(seq++);

 this.sens=0;

 if(estVide(x,y)){

 this.x=x;

 this.y=y;}

 else

 {

 // ici soit on cherche une nouvelle position vide ou on termine l'exécution

 }

}

// cette méthode retourne true si la position es vide (aucun robot

// à cette position), false sinon (un robot occupe la position

public static boolean estVide(int a,int b){

 for(int i=0;i<seq-1;i++)

 if ((robots[i].getX()==a)&& (robots[i].getY()==b)){

 return false;

 }

 return true;

}

public int getX(){return this.x;} // accesseur de x

public int getY(){return this.y;} // accesseur de y

public void avancer(){ // avant d’avancer, on test si la future position est vide

 switch(sens){

 case 0: {if(estVide(x+1,y)) x++;

 break;}

 case 1: {if(estVide(x,y+1)) y++;

 break;}

 case 2: {if(estVide(x-1,y)) x--;

 break;}

 case 3: {if(estVide(x,y-1)) y--;

 break;}

 }

}

public void tourner(){

 sens=(sens+1)%4;

}

public void afficherEtat(){

 System.out.println(nom+":"+x+","+y+"Sens:"+sens);

}

}