Durée : 1h30 Le 21/02/2013

CONTRÔLE FINAL
MASTER II - SRI

Questions de cours (5 pts)

1. Quelles sont les limites des SGBD qui nous obligent de passer aux SGBD orientés objets ? (1 pt)
2. Quelle est l'utilité du module « Bases de données orientées objets » dans le programme de la formation « Systèmes de recherche d'informations » ? (1 pt)
3. Définissez la caractéristique d'extensibilité, et dites si un SGBDOO est extensible. (1 pt)
4. Un objet composant peut être, ou non, partagé entre plusieurs objets de la même classe composite ou bien partagé entre les classes composites ? (1 pt)
5. Dans une base de données orientée objets modélisant l'université de Batna, il y a une classe « Personne » qui est la classe mère d'autres classes (exp : Etudiant, Enseignant, …). Est-ce que les deux classes « Etudiant » et « Enseignant » constituent une couverture de la classe « Personne », et pourquoi ? (1 pt)

Requêtes OQL (6.5 pts)

Dans un système de gestion des vols d'une compagnie aérienne, une classe appelée « Vol » est définie comme suit :

```
public class Vol {
      int Num; //Numéro du vol
      String Origine; //Ville d'origine
      String Destination; //Ville de destination
      Date D_Depart; //Date de départ
      Date D_Arrivee; //Ville d'arrivée
      Time H_Depart; //Heure de départ
      int Duree; //Durée du vol en minutes
}
```

Donnez les requêtes OQL qui répondent aux questions suivantes :

1. L'ensemble des vols. (1 pt)
2. Les vols qui ont la ville de départ « Batna ». (1 pt)
3. Les villes de destination des vols qui ont la ville de départ « Batna » et une durée de 01 heure au plus. (1.5 pt)
4. L'ensemble des villes de destination, de tous les vols, qui ne sont pas des villes d'origine pour d'autres vols. (1.5 pts)
5. Partitionner l'ensemble des vols selon leurs dates de départ en donnant la taille de chaque partition. (1.5 pts)

Modélisation (8.5 pts)

Soit un système de gestion de feux tricolores manipulant des informations sur les intersections entre les rues.

- Une intersection est caractérisée par un identifiant, une adresse, 1er feu, 2ème feu, 3ème feu, 4ème feu. Le 1er feu est en face du 3ème, et le 2ème feu est en face du 4ème.
- Un feu est caractérisé par un identifiant et un état (1 : Rouge, 2 : Orange, 3 : Vert, 4 : Hors service).

1. Ecrivez en java les classes nécessaires pour modéliser la gestion des feux tricolores. La classe principale sera nommée « **GEST_FEUX** ». (0.5 pts)
2. Ecrivez la méthode **getLastId_Feu()** qui retourne le plus grand identifiant des feux enregistrés dans la base. (1 pt)
3. Ecrivez la méthode **getFeuxHS()** qui retourne la liste des feux qui ont un état « Hors service ». (1 pt)
4. Ecrivez la méthode **getIntersectionsHS()** qui retourne la liste des intersections qui ont au-moins un feu « Hors service ». (1 pt)
5. Pour une intersection, .écrivez la méthode **changeEtat()** qui permet de changer les états des feux de l'intersection suivant l'état actuel. (1 pt)
6. Pour un feu hors service d'une intersection, écrivez la méthode **retablirEtat()** qui reçoit en paramètre le numéro du feu dans l'intersection {exp : 1,2,3 ou 4} et permet de rétablir son état selon les états des feux de la même intersection. (1 pt)
7. Ecrivez la méthode **main()** de la classe **GEST_FEUX** qui fait les travaux suivants :
    a. Créer 10 intersections avec 40 feux. Les états des feux d'une intersection seront initialisés d'une manière à ce les feux 1 et 3 seront verts, et les feux 2 et 4 seront rouges. Les adresses des intersections seront initialisées à "adr_i". Tout sera stocké dans une base de données orientée objets. (1 pt)
    b. Afficher pour chaque intersection son identifiant, son adresse et les identifiants des feux avec leurs états. (1 pt)
    c. Changer l'état de l'intersection qui a l'identifiant 1 et afficher son état. (1 pt)

Bonne chance…

NB: Le corrigé type vous le trouverez sur les sites :
http://www.larbiguezouli.com

Correction du contrôle final
Master II - SRI

Questions de cours (5 pts)

1. Quelles sont les limites des SGBD qui nous obligent de passer aux SGBD orientés objets ? (1 pt)
2. Quelle est l'utilité du module « Bases de données orientées objets » dans le programme de la formation « Systèmes de recherche d'informations » ? (1 pt)
3. Définissez la caractéristique d'extensibilité, et dites si un SGBDOO est extensible. (1 pt)
4. Un objet composant peut être, ou non, partagé entre plusieurs objets de la même classe composite ou bien partagé entre les classes composites ? (1 pt)
5. Dans une base de données orientée objets modélisant l'université de Batna, il y a une classe « Personne » qui est la classe mère d'autres classes (exp : Etudiant, Enseignant, …). Est-ce que les deux classes « Etudiant » et « Enseignant » constituent une couverture de la classe « Personne », et pourquoi ? (1 pt)

Réponse

1. **Les limites des SGBD :**

- **Le modèle de données simple ne représente pas facilement les objets du monde réel.**
- **L'incompatibilité des LMD relationnels et des langages de programmation.**
- **Le développement d'applications est très lent et les applications sont difficilement maintenables.**

2. **Les données du monde réel traitées dans le domaine de la Recherche d'Information sont de grand volume et leurs contenu est hétérogène ce qui nous dirige de représenter ces données sous forme d'objets.**
3. **L'extensibilité : supporter la possibilité d'ajouter de nouveaux types. Un SGBDOO doit être extensible afin de prendre en compte de nouveaux domaines d'application.**
4. **Un objet composant peut être, ou non, partagé entre plusieurs objets de la même classe composite.**
5. **Non, parce que on peut trouver un objet de la classe « Personne » qui n'est ni « Etudiant » ni « Enseignant » comme par exemple « Comptable ».**
   Alors que la définition de la couverture est comme suit : « Un sous-ensemble S de classes spécifiques immédiates d'une classe générique C est dit couvrant si toute occurrence de C est aussi occurrence d'au moins une des classes de S ».

Requêtes OQL (6.5 pts)

Dans un système de gestion des vols d'une compagnie aérienne, une classe appelée « Vol » est définie comme suit :

```
public class Vol {
      int Num; //Numéro du vol
      String Origine; //Ville d'origine
      String Destination; //Ville de destination
      Date D_Depart; //Date de départ
```

```
        Date D_Arrivee; //Ville d'arrivée
        Time H_Depart; //Heure de départ
        int Duree; //Durée du vol en minutes
    }
```

Donnez les requêtes OQL qui répondent aux questions suivantes :

1. L'ensemble des vols. (1 pt)
2. Les vols qui ont la ville de départ « Batna ». (1 pt)
3. Les villes de destination des vols qui ont la ville de départ « Batna » et une durée de 01 heure au plus. (1.5 pt)
4. L'ensemble des villes de destination, de tous les vols, qui ne sont pas des villes d'origine pour d'autres vols. (1.5 pts)
5. Partitionner l'ensemble des vols selon leurs dates de départ en donnant la taille de chaque partition. (1.5 pts)

Réponse

1. **La requête OQL qui retourne l'ensemble des vols : Vol**
2. **SELECT DISTINCT v FROM v IN Vol WHERE v.Origine="Batna"**
3. **SELECT DISTINCT v.Destination FROM v IN Vol WHERE v.Origine="Batna" AND v.Duree ≤ 60**
4. **SELECT DISTINCT v.Destination FROM v IN Vol WHERE v.Destination NOT IN (SELECT v1.Origine FROM v1 IN Vol)**
5. **GROUP v IN Vol BY (dd : v.D_Depart) WITH (nbVols : COUNT(partition))**

Modélisation (8.5 pts)

Soit un système de gestion de feux tricolores manipulant des informations sur les intersections entre les rues.

- Une intersection est caractérisée par un identifiant, une adresse, 1er feu, 2ème feu, 3ème feu, 4ème feu. Le 1er feu est en face du 3ème, et le 2ème feu est en face du 4ème.
- Un feu est caractérisé par un identifiant et un état (1 : Rouge, 2 : Orange, 3 : Vert, 4 : Hors service).

1. Ecrivez en java les classes nécessaires pour modéliser la gestion des feux tricolores. La classe principale sera nommée « **GEST_FEUX** ». (0.5 pts)
2. Ecrivez la méthode **getLastId_Feu()** qui retourne le plus grand identifiant des feux enregistrés dans la base. (1 pt)
3. Ecrivez la méthode **getFeuxHS()** qui retourne la liste des feux qui ont un état « Hors service ». (1 pt)
4. Ecrivez la méthode **getIntersectionsHS()** qui retourne la liste des intersections qui ont au-moins un feu « Hors service ». (1 pt)
5. Pour une intersection, .écrivez la méthode **changeEtat()** qui permet de changer les états des feux de l'intersection suivant l'état actuel. (1 pt)
6. Pour un feu hors service d'une intersection, écrivez la méthode **retablirEtat()** qui reçoit en paramètre le numéro du feu dans l'intersection {exp : 1,2,3 ou 4} et permet de rétablir son état selon les états des feux de la même intersection. (1 pt)
7. Ecrivez la méthode **main()** de la classe **GEST_FEUX** qui fait les travaux suivants :
    a. Créer 10 intersections avec 40 feux. Les états des feux d'une intersection seront initialisés d'une manière à ce les feux 1 et 3 seront verts, et les feux

2 et 4 seront rouges. Les adresses des intersections seront initialisées à "adr¡". Tout sera stocké dans une base de données orientée objets. (1 pt)

b. Afficher pour chaque intersection son identifiant, son adresse et les identifiants des feux avec leurs états. (1 pt)

c. Changer l'état de l'intersection qui a l'identifiant 1 et afficher son état. (1 pt)

Réponse

```
---------------------- La classe FEU ------------------------------------
public class FEU {
	int id;
	int id_intersection;
	int etat;

	public FEU(Integer identifiant, Integer idIntersection, Integer
nouveau_etat) {
		id = identifiant.intValue();
		id_intersection = idIntersection.intValue();
		etat = nouveau_etat.intValue();
	}
}

---------------------- La classe INTERSECTION ----------------------------
import java.util.List;
import com.db4o.ObjectContainer;

public class INTERSECTION {
	int id;
	String adresse;
	FEU feu_1;
	FEU feu_2;
	FEU feu_3;
	FEU feu_4;

	public INTERSECTION(int identifiant) {
		id = identifiant;
		adresse = null;
		feu_1 = null;
		feu_2 = null;
		feu_3 = null;
		feu_4 = null;
	}

	public INTERSECTION(int identifiant, String adr, int lastId_Feu) {
		id = identifiant;
		adresse = adr;
		int lastId = lastId_Feu;
		feu_1 = new FEU(lastId+1, id, 3);
		feu_2 = new FEU(lastId+2, id, 1);
		feu_3 = new FEU(lastId+3, id, 3);
		feu_4 = new FEU(lastId+4, id, 1);
	}

	public void changeEtat(ObjectContainer db) {
		if (feu_1.etat == 3/*Orange*/) {
			feu_1.etat = 2;
			feu_3.etat = 2;
		}else {
			if (feu_1.etat == 1/*Rouge*/) {
				feu_1.etat = 3;
				feu_3.etat = 3;
				feu_2.etat = 1;
```

```
                        feu_4.etat = 1;
                  }else if (feu_1.etat == 2/*Vert*/) {
                              feu_1.etat = 1;
                              feu_3.etat = 1;
                              feu_2.etat = 3;
                              feu_4.etat = 3;
                  }
            }
            db.set(this);
      }

      public void retablirEtat(ObjectContainer db, int numFeu) {
            if (numFeu == 1) {
                  feu_1.etat = feu_3.etat;
            }
            if (numFeu == 2) {
                  feu_2.etat = feu_4.etat;
            }
            if (numFeu == 3) {
                  feu_3.etat = feu_1.etat;
            }
            if (numFeu == 4) {
                  feu_4.etat = feu_2.etat;
            }
            db.set(this);
      }

      public void affiche() {
            System.out.println("Etat de l'intersection : "+id+"\n"+
                        " Adresse : "+adresse+"\n"+
                        " FEU_1: id="+feu_1.id+" état="+feu_1.etat+"\n"+
                        " FEU_2: id="+feu_2.id+" état="+feu_2.etat+"\n"+
                        " FEU_3: id="+feu_3.id+" état="+feu_3.etat+"\n"+
                        " FEU_4: id="+feu_4.id+" état="+feu_4.etat);

      }
}

---------------------- La classe GEST_FEUX --------------------------------
import java.io.File;
import java.util.LinkedList;
import com.db4o.Db4o;
import com.db4o.ObjectContainer;
import com.db4o.ObjectSet;

public class GEST_FEUX {
      public static String DBOFILENAME = "d:\\GEST_FEUX.dbo";

      public static int getLastId_Feu(ObjectContainer db) {
            int maxId = 0;
            ObjectSet result = db.get(FEU.class);
            while (result.hasNext()) {
                  FEU feu = (FEU) result.next();
                  int feuId = ((FEU) result.next()).id;
                  if (maxId < feuId) maxId = feuId;
            }
            return maxId;

      }

      public LinkedList getFeuxHS(ObjectContainer db) {
```

```
            LinkedList ret = new LinkedList();
            FEU my_feu = new FEU(null, null, 4); //4: Etat hors service
            ObjectSet result = db.get(my_feu);
            while (result.hasNext()) {
                  FEU feu = (FEU) result.next();
                  ret.add(feu);
            }
            return ret;
      }

      public LinkedList getIntersectionsHS(ObjectContainer db) {
            LinkedList ret = new LinkedList();
            ObjectSet result = db.get(INTERSECTION.class);
            while (result.hasNext()) {
                  INTERSECTION intersection = (INTERSECTION) result.next();
                  if (intersection.feu_1.etat == 4 ||
                        intersection.feu_2.etat == 4 ||
                        intersection.feu_3.etat == 4 ||
                        intersection.feu_4.etat == 4)
                        ret.add(intersection);
            }
            return ret;
      }

      public static void main(String[] args) {
            //Préparation de la base de données objet
            new File(DBOFILENAME).delete();
            ObjectContainer db = Db4o.openFile(DBOFILENAME);

            //Création de 10 intersections avec 40 feux
            int lastIdFeu = 0;
            for (int i=1; i<=10; i++) {
                  lastIdFeu = getLastId_Feu(db);
                  INTERSECTION intersection = new INTERSECTION(i, "adr"+i,
lastIdFeu);
                  db.set(intersection);
            }

            //Affichage
            ObjectSet result = db.get(INTERSECTION.class);
            while (result.hasNext()) {
                  INTERSECTION intersection = (INTERSECTION) result.next();
                  intersection.affiche();
            }

            INTERSECTION intersection = new INTERSECTION(1);
            result = db.get(intersection);
            if ( result.hasNext() ) {
                  intersection = (INTERSECTION) result.next();
                  intersection.changeEtat(db);
                  intersection.affiche();
            }
            db.close();
      }
}
```

Le code source de la partie modélisation peut être téléchargé à partir du lien suivant :
http://www.larbiguezouli.com/Fichiers/Enseignement/MasterSRI/bdoo/Controle_Final_Correction_2012_2013_Modelisation.zip