Durée : 1h30 Le 04/02/2016

Contrôle final
Bases de données orientées objets

Questions de cours (5 pts)

1. Quelques SGBDO acceptent la contrainte inverse. Définissez cette contrainte.
2. L'héritage multiple est la cause d'un conflit dans les SGBDO. Quelles solutions proposent les SGBDO pour résoudre ce conflit ?
3. Donnez le résultat de cette requête OQL :

```
GROUP x IN (    SELECT e FROM e IN Enseignants
                WHERE EXISTS c IN e.cours-assurés : c.cycle=3
           )
BY ( statut : x.statut )
WITH ( nbr : COUNT(partition) ) ;.
```

4. Modélisons une entreprise en utilisant une base de données orientée objets. Soit la classe « Personne » qui est la classe mère d'autres classes (exp : Développeurs, Commerciaux, Chefs_Projets, …). Est-ce que les deux classes « Développeurs » et « Commerciaux » constituent une couverture de la classe « Personne », et pourquoi ?
5. Définissez l'agrégation.

Modélisation (15 pts)

Un responsable d'un magasin voulait gérer son stock on utilisant une base de données orientée objet.

Pour cela, il enregistre dans la base des informations sur les produits, les ventes et les achats du magasin.

Il enregistre pour chaque produit une désignation, une quantité, un prix d'achat et un prix de vente.

Pour une opération de vente il enregistre un identifiant, une date, le nom du produit vendu et la quantité vendue.

Pour une opération d'achat il enregistre un identifiant, une date, le nom du produit acheté et la quantité achetée.

Quand une opération de vente ou d'achat se fait, la base de données doit être mise à jour.

1. Ecrivez en java les classes nécessaires, avec leurs attributs, pour modéliser le système de gestion de stock de ce magasin. La classe principale sera nommée « **GEST_STOCK** ».
2. Ecrivez un constructeur pour la classe des produits qui prend en argument le nom du produit, la quantité existante et le prix unitaire d'achat. Le prix de vente se calcule en ajoutant 20% au prix d'achat.
3. Ecrivez la méthode **existsInStock()** qui permet de vérifier si une quantité d'un produit est disponible dans le stock ou pas. Elle prend en argument le nom du produit et la quantité demandée.

4. Ecrivez la méthode **buyProduct()** qui simule l'approvisionnement du magasin par un produit. Le gérant achète un produit pour le mettre en vente dans le magasin. Cette méthode permet de mettre à jour la quantité du produit s'il existe dans la base, ou l'ajouter à la base s'il n'existe pas. Elle prend en argument le nom du produit, la quantité achetée, la date d'achat et le prix unitaire d'achat.
5. Ecrivez la méthode **saleProduct()** qui simule la vente d'un produit. Cette méthode permet de mettre à jour la quantité du produit dans la base. Elle prend en argument le nom du produit, la date de vente et la quantité vendue.
6. Ecrivez la méthode **commande()** qui simule une commande d'un client contenant l'achat de plusieurs produits. Cette méthode permet de faire passer une commande d'un client en mettant à jour la base selon la commande du client. Elle prend en argument une liste de ventes.
7. Ecrivez la méthode **afficheStock()** qui permet d'afficher la liste des produits existants dans le stock.

Remarque : Pour les méthodes précédentes, vous devez choisir les paramètres nécessaires et les classes adéquates.

8. Ecrivez la méthode **main()** de la classe **GEST_STOCK** qui fait les travaux suivants :
   a. Acheter 10 produits {P1, …, P10} avec des quantités = 100 et des prix = 100.
   b. Afficher le contenu du stock.
   c. Passez une commande d'un client contenant trois produits :
      i. Produit P1 avec quantité de 5 ;
      ii. Produit P5 avec quantité de 50 ;
      iii. Produit P11 avec quantité de 10.
   d. Afficher le contenu du stock.

Bonne chance…

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

Correction du contrôle final
Bases de données orientées objets

Questions de cours (5 pts)

1. Quelques SGBDO acceptent la contrainte inverse. Définissez cette contrainte.
   L'objet composite dépend de ses objets composants.
2. L'héritage multiple est la cause d'un conflit dans les SGBDO. Quelles solutions proposent les SGBDO pour résoudre ce conflit ?
   - **soit ils refusent la définition d'un tel héritage;**
   - **soit ils demandent au concepteur de choisir la sur-classe «dominante» dont la sous-classe héritera;**
   - **soit ils appliquent une règle (par exemple la première classe citée dans la clause Is-a).**
3. Donnez le résultat de cette requête OQL :

```
GROUP x IN (    SELECT e FROM e IN Enseignants
                WHERE EXISTS c IN e.cours-assurés : c.cycle=3
           )
BY ( statut : x.statut )
WITH ( nbr : COUNT(partition) ) ;.
```

   Cette requête donne en résultat un ensemble de structures:
   SET (STRUCT(statut:STRING, nbr:INT))
   qui définissent le statut de chaque enseignant (parmi les enseignants qui donnent au moins un cours de 3ème cycle) et le nombre d'enseignants de chaque groupe.
4. Modélisons une entreprise en utilisant une base de données orientée objets. Soit la classe « Personne » qui est la classe mère d'autres classes (exp : Développeurs, Commerciaux, Chefs_Projets, …). Est-ce que les deux classes « Développeurs » et « Commerciaux » constituent une couverture de la classe « Personne », et pourquoi ?
   Non, parce que on peut trouver un objet de la classe « Personne » qui n'est ni « Développeur » ni « Commercial » comme par exemple un « Comptable » ou « Chef_Projets ».
   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 ».
5. Définissez l'agrégation.
   C'est une relation entre les objets qui permet de décrire un objet par les objets qui le composent.

Modélisation (15 pts)

Un responsable d'un magasin voulait gérer son stock on utilisant une base de données orientée objet.

Pour cela, il enregistre dans la base des informations sur les produits, les ventes et les achats du magasin.

Il enregistre pour chaque produit une désignation, une quantité, un prix d'achat et un prix de vente.

Pour une opération de vente il enregistre un identifiant, une date, le nom du produit vendu et la quantité vendue.

Pour une opération d'achat il enregistre un identifiant, une date, le nom du produit acheté et la quantité achetée.

Quand une opération de vente ou d'achat se fait, la base de données doit être mise à jour.

1. Ecrivez en java les classes nécessaires, avec leurs attributs, pour modéliser le système de gestion de stock de ce magasin. La classe principale sera nommée « **GEST_STOCK** ». (2pts)
2. Ecrivez un constructeur pour la classe des produits qui prend en argument le nom du produit, la quantité existante et le prix unitaire d'achat. Le prix de vente se calcule en ajoutant 20% au prix d'achat. (1pt)
3. Ecrivez la méthode **existsInStock()** qui permet de vérifier si une quantité d'un produit est disponible dans le stock ou pas. Elle prend en argument le nom du produit et la quantité demandée. (2pts)
4. Ecrivez la méthode **buyProduct()** qui simule l'approvisionnement du magasin par un produit. Le gérant achète un produit pour le mettre en vente dans le magasin. Cette méthode permet de mettre à jour la quantité du produit s'il existe dans la base, ou l'ajouter à la base s'il n'existe pas. Elle prend en argument le nom du produit, la quantité achetée, la date d'achat et le prix unitaire d'achat. (2pts)
5. Ecrivez la méthode **saleProduct()** qui simule la vente d'un produit. Cette méthode permet de mettre à jour la quantité du produit dans la base. Elle prend en argument le nom du produit, la date de vente et la quantité vendue. (2pts)
6. Ecrivez la méthode **commande()** qui simule une commande d'un client contenant l'achat de plusieurs produits. Cette méthode permet de faire passer une commande d'un client en mettant à jour la base selon la commande du client. Elle prend en argument une liste de ventes. (2pts)
7. Ecrivez la méthode **afficheStock()** qui permet d'afficher la liste des produits existants dans le stock. (2pts)

Remarque : Pour les méthodes précédentes, vous devez choisir les paramètres nécessaires et les classes adéquates.

8. Ecrivez la méthode **main()** de la classe **GEST_STOCK** qui fait les travaux suivants : (2pts)
   a. Acheter 10 produits {P1, …, P10} avec des quantités = 100 et des prix = 100.
   b. Afficher le contenu du stock.
   c. Passez une commande d'un client contenant trois produits :
      i. Produit P1 avec quantité de 5 ;
      ii. Produit P5 avec quantité de 50 ;
      iii. Produit P11 avec quantité de 10.
   d. Afficher le contenu du stock.

Solution :

```
public class PRODUIT {
      public String designation;
      public int quantite;
      public double prixAchat;
      public double prixVente;

      public PRODUIT(){
```

```
	}

	public PRODUIT(String nomProduit, int quantiteAchat, double prix) {
		designation = nomProduit;
		quantite = quantiteAchat;
		prixAchat = prix;
		prixVente = prix * 1.2;
	}
}

public class OPERATION {
	public int id;
	public Date date;
	public String designation;
	public int quantite;

}

public class VENTE extends OPERATION {
	public VENTE(ObjectContainer db, Date d, String nomProduit, int
quantiteVendue) {
		date = d;
		designation = nomProduit;
		quantite = quantiteVendue;
		id = getLastId(db) + 1;
	}

	public int getLastId(ObjectContainer db) {
		int lastId = 0;
		ObjectSet<VENTE> ret = db.query(VENTE.class);
		VENTE obj = null;
		for (int i=1; i<=ret.size(); i++) {
			obj = (VENTE) ret.next();
			if (obj.id > lastId) lastId = obj.id;
		}
		return lastId;
	}
}

public class ACHAT extends OPERATION {

	public ACHAT(ObjectContainer db, Date d, String nomProduit, int
quantiteAchat) {
		date = d;
		designation = nomProduit;
		quantite = quantiteAchat;
		id = getLastId(db) + 1;
	}

	public int getLastId(ObjectContainer db) {
		int lastId = 0;
		ObjectSet<ACHAT> ret = db.query(ACHAT.class);
		ACHAT obj = null;
		for (int i=1; i<=ret.size(); i++) {
			obj = (ACHAT) ret.next();
			if (obj.id > lastId) lastId = obj.id;
		}
		return lastId;
	}
```

```
}

public class GEST_STOCK {
	public static String DBOFILENAME = "d:/gest_stock.dbo";

	public static boolean existsInStock(ObjectContainer db, String
nomProduit, int quantite) {
		boolean ret = false;
		PRODUIT myobj = new PRODUIT();
		myobj.designation = nomProduit;
		ObjectSet<PRODUIT> result = db.queryByExample(myobj);
		for (int i=1; i<=result.size(); i++) {
			myobj = result.next();
			if (myobj.quantite >= quantite) ret = true;
		}

		return ret;
	}

	public static void buyProduct(ObjectContainer db, String nomProduit,
int quantite, Date dateAchat, double prix){
		if ( existsInStock(db, nomProduit, 0) ) {
			PRODUIT myPro = new PRODUIT();
			myPro.designation = nomProduit;
			ObjectSet<PRODUIT> result = db.queryByExample(myPro);
			myPro = result.next();
			myPro.quantite += quantite;
			db.store(myPro);
		} else {
			PRODUIT myPro = new PRODUIT(nomProduit, quantite, prix);
			db.store(myPro);
		}
		ACHAT myAchat = new ACHAT(db, dateAchat, nomProduit, quantite);
		db.store(myAchat);
	}

	public static void saleProduct(ObjectContainer db, String nomProduit,
int quantite, Date dateAchat){
		if ( existsInStock(db, nomProduit, quantite) ) {
			PRODUIT myPro = new PRODUIT();
			myPro.designation = nomProduit;
			ObjectSet<PRODUIT> result = db.queryByExample(myPro);
			myPro = result.next();
			myPro.quantite -= quantite;
			if (myPro.quantite > 0) db.store(myPro);
			else db.delete(myPro);
		} else {
			System.out.println("Le produit n'existe pas ou la
quantité du produit en stock n'est pas suffisante!");
		}
	}

	public static void commande(ObjectContainer db, ArrayList<VENTE>
commandeClient) {
		Iterator<VENTE> i = commandeClient.iterator();
		while (i.hasNext()) {
			VENTE obj = i.next();
			saleProduct(db, obj.designation, obj.quantite, obj.date);
		}
	}
```

```
	public static void afficheStock(ObjectContainer db) {
		ObjectSet<PRODUIT> result = db.query(PRODUIT.class);
		for (int i=1; i<result.size(); i++) {
			PRODUIT obj = result.next();
			System.out.println("Produit: " + obj.designation +
					" quantité: " + obj.quantite +
					" prix d'achat: " + obj.prixAchat +
					" prix de vente: " + obj.prixVente);
		}
	}

	public static void main(String[] args) {
		new File(DBOFILENAME).delete();
		ObjectContainer db = Db4o.openFile(DBOFILENAME);
		try {
			for (int i = 1; i <= 10; i++) {
				buyProduct(db, "P"+i, 100, new Date(), 100.0);
			}
			afficheStock(db);

			VENTE v1 = new VENTE(db, new Date(), "P1", 5);
			VENTE v2 = new VENTE(db, new Date(), "P5", 50);
			VENTE v3 = new VENTE(db, new Date(), "P11", 10);
			ArrayList<VENTE> commandeClient = new ArrayList<VENTE>();
			commandeClient.add(v1);
			commandeClient.add(v2);
			commandeClient.add(v3);
			commande(db, commandeClient);
			afficheStock(db);
		} finally {
			db.close();
		}
	}
}
```

Pour télécharger la solution de cette partie cliquez sur ce lien :
http://larbiquezouli.com/images/Enseignement/MasterSRI/BDOO/Controle_Final_Correction_2015_2016_Modelisation.zip