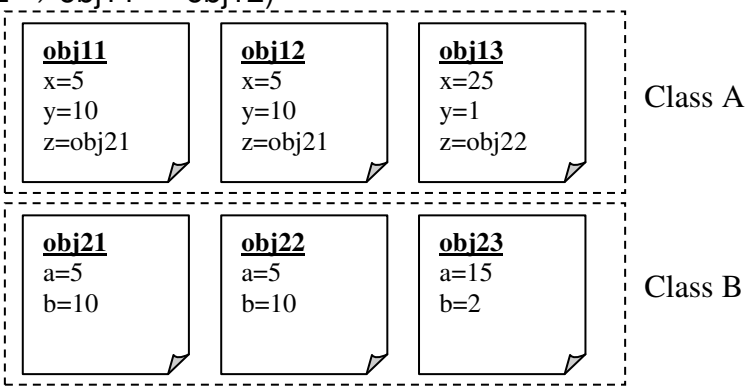
Durée : 1h30 Le 09/01/2014

CONTRÔLE FINAL
MASTER II - SRI

Questions de cours (5 pts)

1. Définissez l'agrégation.
2. Quand est-ce qu'on dit qu'un objet *x* est dépendant d'un objet père *y* ?
3. Parmi les expressions suivantes, lesquelles sont vrai et lesquelles sont fausses :

- obj11 == obj12
- obj11.z == obj12.z
- obj11 =1 obj12
- obj11 =1 obj13
- obj11 =* obj13
- obj21 =1 obj22
- obj21 == obj22
- (obj11 =1 obj12 ⇒ obj11 =* obj12)



4. Quelle est le problème causé par l'héritage multiple ?

Requêtes OQL (6 pts)

Dans un système de gestion de comptes d'une banque, une classe appelée « COMPTE » est définie comme suit :

```
public class COMPTE {
      int Num; //Numéro du compte
      int NumClient; //Numéro du client
      Date D_Ouverture; //Date d'ouverture du compte
      double Solde; //Solde actuel du compte
}
```

Donnez les requêtes OQL qui répondent aux questions suivantes :

1. L'ensemble des comptes de la banque. (1 pt)
2. Les comptes du client numéro 10. (1 pt)
3. Les comptes ouverts après 01/01/2000. (1 pt)
4. Les comptes créditeurs. (1 pt)
5. Partitionner l'ensemble des comptes selon leurs dates d'ouverture en donnant la taille de chaque partition. (2 pts)

Modélisation (9 pts)

Soit un système de gestion d'un parking à étages. Chaque étage possède une capacité en nombre de voitures.

- Un parking est caractérisé par un identifiant et un nombre d'étages.
- Un étage est caractérisé par un identifiant, une capacité et le nombre de places libres.

1. Ecrivez en java les classes nécessaires, avec leurs attributs, pour modéliser la gestion du parking. La classe principale sera nommée « **PARKING** ». (0.5 pt)
2. Ecrivez la méthode **isFloorFull()** qui permet de vérifier si un étage est plein. (0.5 pt)
3. Ecrivez la méthode **getFirstFreeFloor()** qui retourne l'étage le plus bas contenant des places libres. (1 pt)
4. Ecrivez la méthode **getFreePlaces()** qui retourne le nombre de places libres dans le parking. (1 pt)
5. Ecrivez la méthode **isParkingFull()** qui permet de vérifier si le parking est plein. (0.5 pt)
6. Ecrivez les méthodes **carIn()** qui permet de faire entrer un véhicule dans le parking en le plaçant dans l'étage le plus bas. (1 pt)
7. Ecrivez les méthodes **carOut()** qui permet de faire sortir un véhicule d'un étage du parking. (1 pt)

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

8. Ecrivez la méthode **main()** de la classe **PARKING** qui fait les travaux suivants :
   a. Créer un parking avec 5 étages de 20 places chacun. Les étages sont vides. Tout sera stocké dans une base de données orientée objets. (1 pt)
   b. Faites entrer 25 véhicules au parking en remplissant les étages du plus bas au plus haut. (1 pt)
   c. Faites sortir un véhicule de l'étage le plus bas. (0.5 pt)
   d. Afficher pour chaque étage son identifiant, sa capacité et son nombre de places libres. (1 pts)

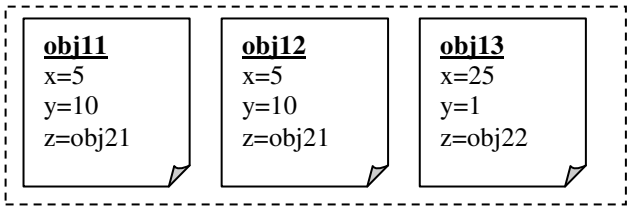
Bonne chance…

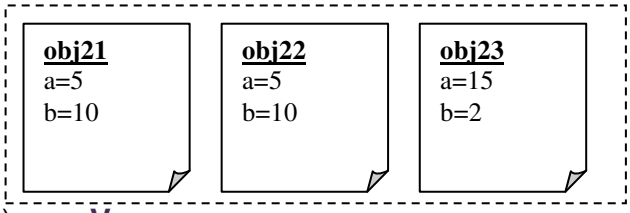
NB: Le corrigé type vous le trouverez sur le site :
http://www.larbiguezouli.com

CORRECTION DU CONTRÔLE FINAL
MASTER II - SRI

Questions de cours (5 pts)

1. Définissez l'agrégation.
   C'est une relation entre les objets qui permet de décrire un objet par les objets qui le composent.
2. Quand est-ce qu'on dit qu'un objet *x* est dépendant d'un objet père *y* ?
   Un objet *x* est dépendant d'un objet père *y* si la destruction de *y* entraîne automatiquement celle de *x*
3. Parmi les expressions suivantes, lesquelles sont vrai et lesquelles sont fausses :





obj11 == obj12 **F**
obj11.z == obj12.z **V**
obj11 =1 obj12 **V**
obj11 =1 obj13 **F**
obj11 =* obj13 **V**
obj21 =1 obj22 **V**
obj21 == obj22 **F**
(obj11 =1 obj12 ⇒ obj11 =* obj12) **V**

4. Quelle est le problème causé par l'héritage multiple ? (1 pt)
   Un conflit peut survenir lors de l'héritage multiple des attributs et des méthodes: deux attributs ou méthodes de deux surclasses peuvent avoir le même nom.

Requêtes OQL (6 pts)

Dans un système de gestion de comptes d'une banque, une classe appelée « COMPTE » est définie comme suit :

```
public class COMPTE {
        int Num; //Numéro du compte
        int NumClient; //Numéro du client
        Date D_Ouverture; //Date d'ouverture du compte
        double Solde; //Solde actuel du compte
}
```

Donnez les requêtes OQL qui répondent aux questions suivantes :

1. L'ensemble des comptes de la banque. (1 pt)
2. Les comptes du client numéro 10. (1 pt)
3. Les comptes ouverts après 01/01/2000. (1 pt)
4. Les comptes créditeurs. (1 pt)
5. Partitionner l'ensemble des comptes selon leurs dates d'ouverture en donnant la taille de chaque partition. (2 pts)

Réponse

1. **La requête OQL qui retourne l'ensemble des comptes de la banque : COMPTE**
2. **SELECT DISTINCT c FROM c IN COMPTE WHERE c.NumClient=10**

3. **SELECT DISTINCT c FROM c IN COMPTE WHERE c.D_Ouverture > '01/01/2000'**
4. **SELECT DISTINCT c FROM c IN COMPTE WHERE c.Solde<0**
5. **GROUP c IN COMPTE BY (dd : c.D_Ouverture) WITH (nbCpts : COUNT(partition))**

Modélisation (9 pts)

Soit un système de gestion d'un parking à étages. Chaque étage possède une capacité en nombre de voitures.

- Un parking est caractérisé par un identifiant et un nombre d'étages.
- Un étage est caractérisé par un identifiant, une capacité et le nombre de places libres.

1. Ecrivez en java les classes nécessaires, avec leurs attributs, pour modéliser la gestion du parking. La classe principale sera nommée « **PARKING** ». (0.5 pt)
2. Ecrivez la méthode **isFloorFull()** qui permet de vérifier si un étage est plein. (0.5 pt)
3. Ecrivez la méthode **getFirstFreeFloor()** qui retourne l'étage le plus bas contenant des places libres. (1 pt)
4. Ecrivez la méthode **getFreePlaces()** qui retourne le nombre de places libres dans le parking. (1 pt)
5. Ecrivez la méthode **isParkingFull()** qui permet de vérifier si le parking est plein. (0.5 pt)
6. Ecrivez les méthodes **carIn()** qui permet de faire entrer un véhicule dans le parking en le plaçant dans l'étage le plus bas. (1 pt)
7. Ecrivez les méthodes **carOut()** qui permet de faire sortir un véhicule d'un étage du parking. (1 pt)

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

8. Ecrivez la méthode **main()** de la classe **PARKING** qui fait les travaux suivants :
   a. Créer un parking avec 5 étages de 20 places chacun. Les étages sont vides. Tout sera stocké dans une base de données orientée objets. (1 pt)
   b. Faites entrer 25 véhicules au parking en remplissant les étages du plus bas au plus haut. (1 pt)
   c. Faites sortir un véhicule de l'étage le plus bas. (0.5 pt)
   d. Afficher pour chaque étage son identifiant, sa capacité et son nombre de places libres. (1 pts)

Réponse

```
public class ETAGE {

	int id;
	int capacite;
	int nb_places_libres;

	//Permet de vérifier si un étage est plein
	public boolean isFloorFull() {
		return (nb_places_libres == 0);
	}
}

public class PARKING {
```

```
	public static String DBOFILENAME = "c:/parking.dbo"; //Nom du fichier
contenant la base de données orientée objet

	int id;
	static int nb_etages;

	//Retourne l'étage le plus bas contenant des places libres
	public static ETAGE getFirstFreeFloor(ObjectContainer db) {
		ETAGE myEtage = null;
		ObjectSet<ETAGE> result = db.query(ETAGE.class);
		for (int i=0; i<result.size(); i++) {
			myEtage = result.next();
			if (!myEtage.isFloorFull()) return myEtage;
		}

		return null;
	}

	//Retourne le nombre de places libres dans le parking
	public int getFreePlaces(ObjectContainer db) {
		int ret = 0;
		ETAGE myEtage = null;
		ObjectSet<ETAGE> result = db.query(ETAGE.class);
		for (int i=1; i<=result.size(); i++) {
			myEtage = result.next();
			ret = ret + myEtage.nb_places_libres;
		}
		return ret;
	}

	//Permet de vérifier si le parking est plein
	public boolean isParkingFull(ObjectContainer db) {
		return (getFreePlaces(db) == 0);
	}

	//Permet de faire entrer un véhicule dans le parking
	public static void carIn(ObjectContainer db) {
		ETAGE myEtage = getFirstFreeFloor(db);
		myEtage.nb_places_libres -= 1;
		db.store(myEtage); //mise à jours dans la base de données
	}

	//Permet de faire sortir un véhicule d'un étage du parking
	public static void carOut(ObjectContainer db, int numEtage) {
		ETAGE myEtage = new ETAGE();
		myEtage.id = numEtage;
		ObjectSet<ETAGE> result = db.queryByExample(myEtage);
		if (result.size()>0) {
			myEtage = result.next();
			myEtage.nb_places_libres += 1;
			db.store(myEtage); //mise à jours dans la base de données

		}
	}

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

```
		ObjectContainer db = Db4o.openFile(DBOFILENAME); // Ouverture de la
base de données objet, si elle n'existe pas elle sera créée
		try {
			//Créer un parking avec 5 étages de 20 places chacun.
			//Les étages sont vides.
			//Tout sera stocké dans une base de données orientée objets
			nb_etages = 5;
			ETAGE myEtage = null;
			for (int i=0; i<nb_etages; i++) {
				myEtage = new ETAGE();
				myEtage.id = i;
				myEtage.capacite = 20;
				myEtage.nb_places_libres = 20;
				db.store(myEtage);
			}

			//Faire entrer 25 véhicules au parking
			for (int i=0; i<25; i++) {
				carIn(db);
			}

			//Faites sortir un véhicule du premier étage
			carOut(db, 0);

			//Afficher pour chaque étage son identifiant, sa capacité et
son nombre de places libres.
			myEtage = null;
			ObjectSet<ETAGE> result = db.query(ETAGE.class);
			for (int i=1; i<=result.size(); i++) {
				myEtage = result.next();
				System.out.println("Etage: "+myEtage.id+" - Capacité:
"+myEtage.capacite+" - Places vides: "+myEtage.nb_places_libres);
			}

		} finally {
			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_2013_2014_Modelisation.zip