

Correction Examen POO janvier 2003

Exercice 1 :

Q 1.

Q 1.1. L'attribut de classe (statique) `cpt` permet d'assurer que chaque instance du type énuméré a une valeur pour numéro unique. Ceci est assuré par l'incrémentation "automatique" dans le constructeur.

Q 1.2.

```
public int compareTo(Object o) {
    return this.numero - ((Mois) o).numero; // peut " throw ClassCastException"
}
```

Q 1.3. car par défaut la méthode `equals` compare les adresses, et se comporte donc comme "==" , or dans un "type énuméré", les instances sont `static` et `final` et le constructeur est privé, deux références de type `Mois` seront donc égales ssi elles désignent le même objet, et donc si elles sont "==".

Q 2. Q 3. et Q 4.

```
package util.date;
public class Date {
    private final int jour;
    private final Mois mois;
    private final int annee;
    public Date(int jour, Mois mois, int annee) {
        this.jour = jour; this.mois = mois; this.annee = annee;
    }
    public boolean isBissextile() {
        return (this.annee % 400 == 0) || ((this.annee % 4 == 0) && (this.annee % 100 != 0));
    }
    public int compareTo(Object o) { // méthode rajoutée,
        Date other = (Date) o; // peut " throw ClassCastException"
        if (other.annee != this.annee) return this.annee - other.annee;
        if (other.mois != this.mois) return this.mois.compareTo(other.mois);
        return this.jour - other.jour;
    }
    //===== Version 1
    public int differenceEnJours_v1(Date d) {
        if (this.compareTo(d) < 0)
            return - d.differenceEnJours(this);
        // maintenant on est sûr que d < this
        int result;
        if (d.annee != this.annee) {
            result = d.differenceEnJours(new Date(31, Mois.DECEMBRE, d.annee)) + 1;
            d = new Date(1, Mois.JANVIER, d.annee+1);
        }
        while (d.annee != this.annee) {
            if (d.isBissextile())
                result = result + 366;
            else
                result = result + 365;
            d = new Date(d.jour, d.mois, d.annee+1);
        } // maintenant d.annee == this.annee
        if (d.mois != this.mois) {
            result = result + d.mois.getNbJours(d.isBissextile()) - d.jour + 1;
            d = new Date(1, moisSuivant(this.mois), d.annee);
        }
        while (this.mois != d.mois) {
            result = result + d.mois.getNbJours(d.isBissextile());
            d = new Date(1, moisSuivant(d.mois), d.annee);
        } // maintenant d.mois = this.mois
        result = result + (this.jour - d.jour); // rappel : d < this
    }
}
```

```

        return result;
    }
    private Mois moisSuivant(Mois m) { // devrait être dans classe Mois
        Mois[] lesMois = new Mois[]{Mois.JANVIER, ..., Mois.DECEMBRE};
        return lesMois[(m.getNumero()+1)%12];
    }
    //===== Version 2
    private Date lendemain() {
        if (this.jour == this.mois.getNbJours(this.isBissextile())) {
            if (this.mois != Mois.DECEMBRE) {
                return new Date(1, Mois.JANVIER, this.annee+1);
            } else {
                return new Date(1, moisSuivant(this.mois), this.annee);
            }
        }
        else
            return new Date(this.jour+1, this.mois, this.annee);
    }
    public int differenceEnJours_v2(Date d) {
        if (this.compareTo(d) < 0)
            return - d.differenceEnJours(this);
        // maintenant on est sûr que d < this

        int result = 0;
        while (d.compareTo(this) < 0) {
            d = d.lendemain();
            result++;
        }
        return result;
    }
    //===== Version 3
    private int nbJoursDepuisZero() {
        int result = this.jour;
        for (Mois m=Mois.JANVIER; m!=this.mois;m = this.moisSuivant()) {
            result = result + m.getNbJours(this.isBissextile());
        }
        for (int a = 0; a < this.annee; a++) {
            result = result + 365;
            if (new Date(1,Mois.JANVIER,a).isBissextile()) result = result +1;
        }
        return result;
    }
    public int differenceEnJours_v3(Date d) {
        if (this.compareTo(d) < 0)
            return - d.differenceEnJours(this);
        // maintenant on est sûr que d < this

        return this.nbJoursDepuisZero()-d.nbJoursDepuisZero();
    }
}

```

Exercice 2 :

Q1.

```

package banque.util;
public interface Identite {
    public String toString();
    public boolean equals(Object o);
}

```

```

package banque.util;
public class PersonneId implements Identite {
    private String nom;
    private String prenom;
    public PersonneId(String nom, String prenom) {
        this.nom = nom; this.prenom = prenom;
    }
    public String toString() { return this.nom+" "+this.prenom; }
    public boolean equals(Object o) {

```

```

        if (! (o instanceof PersonneId)) return false;
        PersonneId other = (PersonneId) o;
        return this.nom.equals(other.nom) && this.prenom.equals(other.prenom);
    }
}

```

```

package banque.util;
public class EntrepriseId implements Identite {
    private String raisonSociale;
    private PersonneId responsable;
    public EntrepriseId(String raisonSociale, PersonneId responsable) {
        this.raisonSociale = raisonSociale; this.responsable = responsable;
    }
    public String toString() { return this.raisonSociale; }
    public boolean equals(Object o) {
        if (! (o instanceof EntrepriseId)) return false;
        EntrepriseId other = (EntrepriseId) o;
        return this.raisonSociale.equals(other.raisonSociale)
            && this.responsable.equals(other.responsable);
    }
}

```

Exercice 3 :

Q 1.

banque::compte::Ecriture
- date : util::date::Date - intitule : String - montant : float
+ Ecriture(d : Date, i : String, m : float) + getDate() : util::date::Date + getIntitule() : String + getMontant() : float

Exercice 4 :

Q 1.

```

package banque.compte;
import banque.util.*;
import java.util.*;
public class CompteCheque implements Compte {
    private float autorisationDecouvert;
    public float getautorisationDecouvert() { return this.autorisationDecouvert; }
    public void setautorisationDecouvert(float val) { this.autorisationDecouvert = val; }

    private String numeroCompte;
    private float solde;
    private List ecritures;
    public CompteCheque(String numeroCompte) {
        this.numeroCompte = numeroCompte;
        this.solde = 0;
        this.ecritures = new ArrayList();
        this.autorisationDecouvert = 0;
    }
    public String getNumeroCompte() { return numeroCompte; }
    public float getSolde() { return solde; }
    public List getEcritures() { return ecritures; }
    public void addEcriture(Ecriture e) throws UnsupportedOperationException {
        if (e.getMontant() + this.solde < this.autorisationDecouvert) {
            throw new UnsupportedOperationException("ecriture refusee");
        }
        ecritures.add(e);
        this.solde = this.solde()+e.getMontant();
    }
    public List ecrituresDepuis(util.date.Date d, int nbJours) {
        List result = new ArrayList();
    }
}

```

```

        Iterator it = this.ecritures.iterator();
        while (it.hasNext()) {
            Ecriture e = (Ecriture) it.next();
            int diff = d.differenceEnJours(e.getDate());
            if (diff >=0 && diff <= nbjours) {
                result.add(e);
            }
        }
        return result;
    }
}

```

Exercice 5 :

Q1.

```

package banque.client;
import banque.compte.*;
import java.util.*;
public interface Client {
    public Identite getIdentite();
    public String getAdresse();
    public List getComptes();
    public void ajouteCompte(Compte new);
    public Compte getCompte(String numCpte);
    public void addEcriture(String numCpte,String intitule,
                           Date date, float montant) throws UnsupportedOperationException;
}

```

```

package banque.client;
import banque.compte.*;
import java.util.*;
public interface ClientPhysique implements Client {
    private PersonneId id;
    private String adresse;
    private List comptes;
    public ClientPhysique(PersonneId id, String adresse) {
        this.id = id; this.adresse = adresse; this.comptes = new ArrayList();
    }
    public Identite getIdentite() { return this.id; }
    public String getAdresse() { return this.adresse; }
    public List getComptes() { return this.comptes; }
    public void ajouteCompte(Compte nouveauCompte) {
        this.comptes.add(nouveauCompte);
    }
    public Compte getCompte(String numCpte) {
        Iterator it = comptes.iterator();
        Compte result = null;
        while (it.hasNext() && result == null) {
            Compte c = (Compte) it.next();
            if (c.getNumeroCompte().equals(numCpte))
                result = c;
        }
        return result;
    }
    public void addEcriture(String numCpte,String intitule,
                           Date date, float montant) throws UnsupportedOperationException {
        Ecriture e = new Ecriture(date, intitule, montant);
        Compte c = getCompte(numCpte);
        if (c == null) {
            throw new UnsupportedOperationException("compte invalide");
        }
        c.addEcriture(e);
    }
    public boolean equals(Object o) {
        if (! o instanceof ClientPhysique) return false;
        ClientPhysique other = (ClientPhysique) o;
        return other.id.equals(this.id) && other.adresse.equals(this.adresse);
    }
}

```

```

public int hashCode() { // pour l'exo 6
    return (id.toString()+" "+adresse).hashCode();
}
public float soldeCumule() { // pour exo 6 Q 5
    float result = 0;
    Iterator it = comptes.iterator();
    while (it.hasNext()) {
        result = result + ((Compte) it.next()).getSolde();
    }
    return solde;
}
}

```

Exercice 6 :

```

Q1. Q2. Q3. Q4. et Q5. package banque;
import banque.client.*;
import banque.compte.*;
import java.util.*;
public class Banque {
    private Set clients;
    private Map comptes;
    public Banque() {
        this.clients = new HashSet(); this.comptes = new HashMap();
    }
    public void creeCompteCheque(String numCpte, Client client) {
        CompteCheque cpteChq = new CompteCheque(numCpte);
        client.ajouteCompte(cpteChq);
        this.comptes.put(numCpte,cpteChq);
    }
    public List clientsARisque() {
        List result = new ArrayList();
        Iterator it = clients.iterator();
        while (it.hasNext()) {
            Client client = (Client) it.next();
            Iterator cptIt = client.getComptes().iterator();
            while(cptIt.hasNext()) {
                Compte cpt = (Compte) cptIt.next();
                if (cpt.getSolde() < 0)
                    result.add(client); // et on pourrait quitter la boucle "break" ? :-\
            }
        }
        return result;
    }
    public Client getClient(Compte cpt) {
        Client result = null;
        Iterator it = clients.iterator();
        while (it.hasNext() && result == null) {
            Client client = (Client) it.next();
            if (client.getCompte(cpt.getNumeroCompte()))
                result = client;
        }
        return null;
    }
    public Client[] dixMeilleursClients() { // question débile donc solution débile
        Client[] dixMeilleurs = new Client[10]; // 10 meilleurs triés par ordre croissant
        Iterator it = clients.iterator();
        while (it.hasNext()) {
            int idx = 0;
            Client c = (Client) it.next();
            while ( (idx < dixMeilleurs.length)
                && (dixMeilleurs[idx] != null)
                && (c.soldeCumule() < dixMeilleurs[idx].soldeCumule()) ) {
                dixMeilleurs[idx] = c;
                idx++;
            }
            if (dixMeilleurs[idx] == null) { // pas encore 10 cptes connus
                dixMeilleurs[idx] = c;
            }
        }
    }
}

```

```
    }
    else if (idx < dixMeilleurs.length) {           // on decale pour ranger triés
        for (j = dixMeilleurs.length -1; j > idx; j++) {
            dixMeilleurs[j] = dixMeilleurs[j-1];
        }
        dixMeilleurs[idx] = c;
    }
}
return dixMeilleurs;
}
}
```