Université Abou Beckr Belkaid
Master 2 Génie Logiciel
Année Universitaire 2019/2020

Examen Final

—

Les technique de construction d'architectures logicielles avancées

—

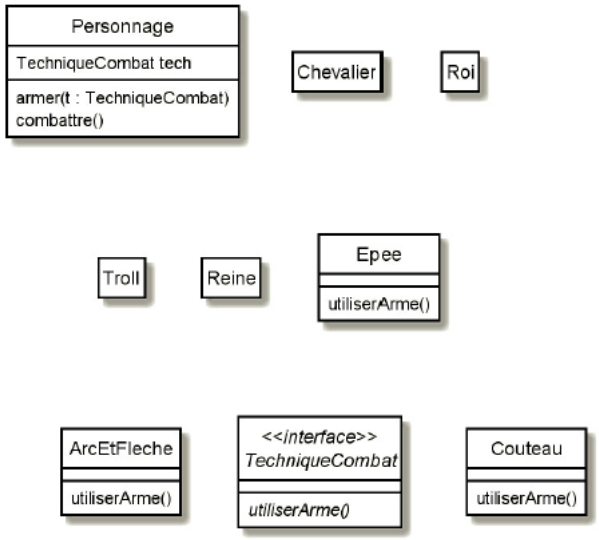
Exercice 1 :

Dans cet exercice, nous allons utilisé le design pattern stratégie.

Questions :

1 Définisse le design pattern stratégie (Sa définition, son diagramme UML et le problème résolu).

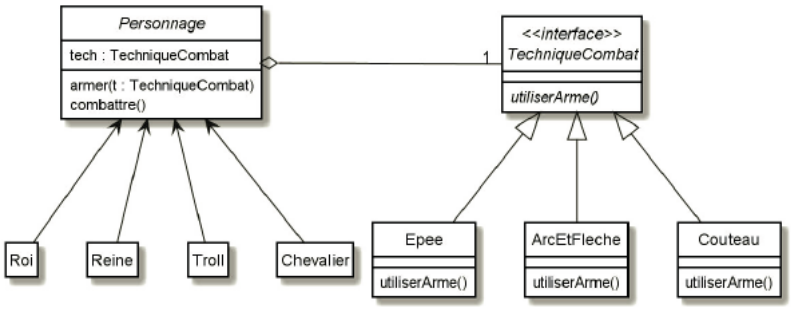
2 En suivant le principe du pattern Stratégie, replacer les classes suivantes dans un schéma UML.



3 Donner en Java l'implémentation de la classe **Personnage**.

Solution :

2



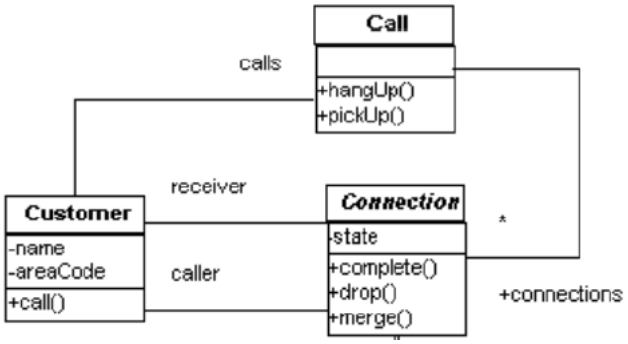
3
```
abstract public class Personnage{
TechniqueCombat tech;
void armer(TechniqueCombat t){tech = t;}
void combattre(){tech.utiliserArme();}
}
```

Exercice 2 :

Nous représentons la simulation d'un système téléphonique classique.
L'utilisateur du téléphone peut :

— Passer un appel.
— Répondre à un appel.
— Rejoindre un appel.
— Couper un appel (Raccrocher).

La conception du système est donnée comme suite :



Les deux classes **Customer** et **Connection** sont données comme suite :

```
public class Customer {

      private String name;
      private int areacode;
      private Vector calls = new Vector();

      protected void removeCall(Call c){
          calls.removeElement(c);
      }

      protected void addCall(Call c){
          calls.addElement(c);
      }

      public Customer(String name, int areacode) {
          this.name = name;
          this.areacode = areacode;
      }

      public String toString() {
          return name + "(" + areacode + ")";
      }

      public int getAreacode(){
          return areacode;
      }

      public boolean localTo(Customer other){
          return areacode == other.areacode;
      }

      public Call call(Customer receiver) {
          Call call = new Call(this, receiver);
          addCall(call);
          return call;
      }

      public void pickup(Call call) {
          call.pickup();
          addCall(call);
      }

      public void hangup(Call call) {
          call.hangup(this);
          removeCall(call);
      }

      public void merge(Call call1, Call call2){
          call1.merge(call2);
          removeCall(call2);
      }
  }
```

```
abstract class Connection {

    public static final int PENDING = 0;
    public static final int COMPLETE = 1;
    public static final int DROPPED = 2;

    Customer caller, receiver;
    private int state = PENDING;

    Connection(Customer a, Customer b) {
        this.caller = a;
        this.receiver = b;
    }

    public int getState(){
        return state;
    }

    public Customer getCaller() { return caller; }

    public Customer getReceiver() { return receiver; }

    void complete() {
        state = COMPLETE;
        System.out.println("connection completed");
    }

    void drop() {
        state = DROPPED;
        System.out.println("connection dropped");
    }

    public boolean connects(Customer c){
        return (caller == c || receiver == c);
    }

}
```

Nous souhaitons ajouter a ce système de nouvelles fonctionnalités pour cela nous proposons l'utilisation de la programmation orientée aspect.

Questions :

1 Donner la définition de la programmation orientée aspect.
2 Ajouter un **minuteur** (Timing), qui permet de compter le temps d'appel de chaque utilisateur.Utiliser l'AOP pour intégrer le minuteur au système téléphonique. Pour cela vous pouvez utiliser la classe **Timing** donnée comme suite :

```
class Timer {
    long startTime, stopTime;

    public void start() {
        startTime = System.currentTimeMillis();
        stopTime = startTime;
    }

    public void stop() {
        stopTime = System.currentTimeMillis();
    }

    public long getTime() {
        return stopTime - startTime;
    }
}
```

Solution :

1 L'AOP est un paradigme de programmation qui permet d'ajouter des fonctionnalités à un code existant sans toucher à ce dernier.

2 Voici le code de l'aspect à ajouter :

Solution 1 :

```
public aspect Timing{
public long Customer.totalConnectTime = 0;
public long getTotalConnectTime(Customer cust){
return cust.totalConnectTime;
}
private Timer Connection.Timer = new Timer();
public Timer getTimer(Connection conn){return conn.timer;}
after (Connection c): target(c) && call(void Connection.complete()){
getTimer(c).start();
}
pointcut endTiming(Connection c):target(c) && call (void Connection.drop());
after (Connection c):endTiming(c){
getTimer(c).stop();
c.getCaller.totalConnectTime +=getTimer(c).getTime();
}
}
```

Solution 2 :

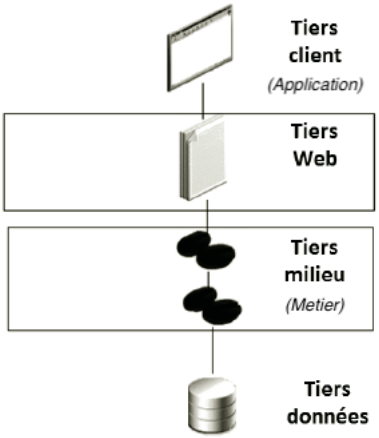
```
public aspect Timing{
public long Customer.totalConnectTime = 0;
public long getTotalConnectTime(Customer cust){
return cust.totalConnectTime;
}
private Timer Customer.Timer = new Timer();
public Timer getTimer(Customer conn){return conn.timer;}
after (Customer c): target(c) && call(void Customer.pickup()){
getTimer(c).start();
}
pointcut endTiming(Customer c):target(c) && call (void Customer.hangup());
after (Customer c):endTiming(c){
getTimer(c).stop();
c.getCaller.totalConnectTime +=getTimer(c).getTime();
}
}
```

Exercice 3 :

Il faut mettre en place une application dédié à la gestion d'un catalogue de livres. L'administrateur de ce ca-

talogue peut : Ajouter un livre, rechercher un livre, supprimer un livre. La base de données se réduit à la seul table Livre donnée comme suite :

Le client souhaite avoir une application web, donnée comme suite :



Questions :

1 Dans quelle couche de l'architecture web se place le conteneur EJB ? Justifier votre réponse.
2 Donner la conception et l'implémentation de la partie EJB de l'application.
3 A chaque ajout de livre, les clients reçoivent un message sous forme de **String**. Améliorer la conception précédente pour ajouter la nouvelle fonctionnalité. Justifier l'utilisation du design pattern approprié.

Réponse :

1 C'est la couche métier qui contient l'implémentation EJB.

2 Donner la conception et l'implémentation de la partie EJB de l'application.

```
@Local
public interface FacadeGestionLiv
   public void ajouter(Livre livr
   public void supprimer(String i
   public Livre rechercherLivre(S
   public List<Livre> afficher(St
}
```

```java
@Stateless
public class FacadeGestionLivresBean implements FacadeGestionLivres {

   @PersistenceContext(name="maBase")
   EntityManager em;

   public void ajouter(Livre livre) {
     em.persist(livre);
   }
   public void supprimer(String id) {
     Livre livre = em.find(Livre.class, id);
     em.remove(livre);
   }
   public Livre rechercherLivre(String id) {
     return em.find(Livre.class, id);
   }
   @SuppressWarnings("unchecked")
   public List<Livre> afficher(String champ, String ordre) {
     Query query = null;
     if (champ==null && ordre==null) {
       query = em.createQuery("SELECT l FROM Livre l");
     } else {
       query = em.createQuery(
       "SELECT l FROM Livre l ORDER BY l."+champ+" "+ordre);
     }
     List<Livre> liste = query.getResultList();
     return liste;
   }
```

.

```java
@Entity
@Table(name = "livre")
public class Livre implements Serializable, Cloneable {
  private String titre;
  private String description;
  private double prix;
  private String auteur;
  @Id
  private String id;

  public Livre() {}
  public String getTitre() {return titre;}
  public void setTitre(String t) {
    titre = t;
  }
  public String getDescription() {return description;}
  public void setDescription(String d) {
    description = d;
  }
  public double getPrix() {return prix;}
  public void setPrix(double p) {
    prix = p;
  }
  public String getAuteur() {return auteur;}
  public void setAuteur(String a) {
    auteur = a;
  }
  public String getId() {return id;}
  public void setId(String i) {
    id = i;
  }
  public String toString() {
    return "Livre [titre=" + titre + ", description=
      description + ", auteur=" + auteur + ", prix=
      prix + ", id=" + id + "]";
  }
  public Object clone() {
    Object o = null;
    try {
      o = super.clone();
    } catch(CloneNotSupportedException e) {}
    return o;
  }
}
```

.

3 Utiliser le design pattern **observer**. La liste des observateurs représente les clients et l'observé c'est l'ajout du livre.

Bon courage et bonne continuation.