CPOA - Support TD 4

Version corrigée 

Cette version comporte des indications pour les réponses aux exercices.

PreReq	1. Je sais programmer en Java. 2. J'ai conscience qu'il faut réfléchir avant de se lancer dans le codage. 3. Je maîtrise patrons de conception. 4. Je maîtrise les diagrammes UML de classe, de séquence et d'états
ObjTD	Aborder quelques subtilités **UML**.
Durée	**1** TD

1. Différences entre dépendance, association, composition, agrégation

Soit le diagramme de classe partiel suivant :

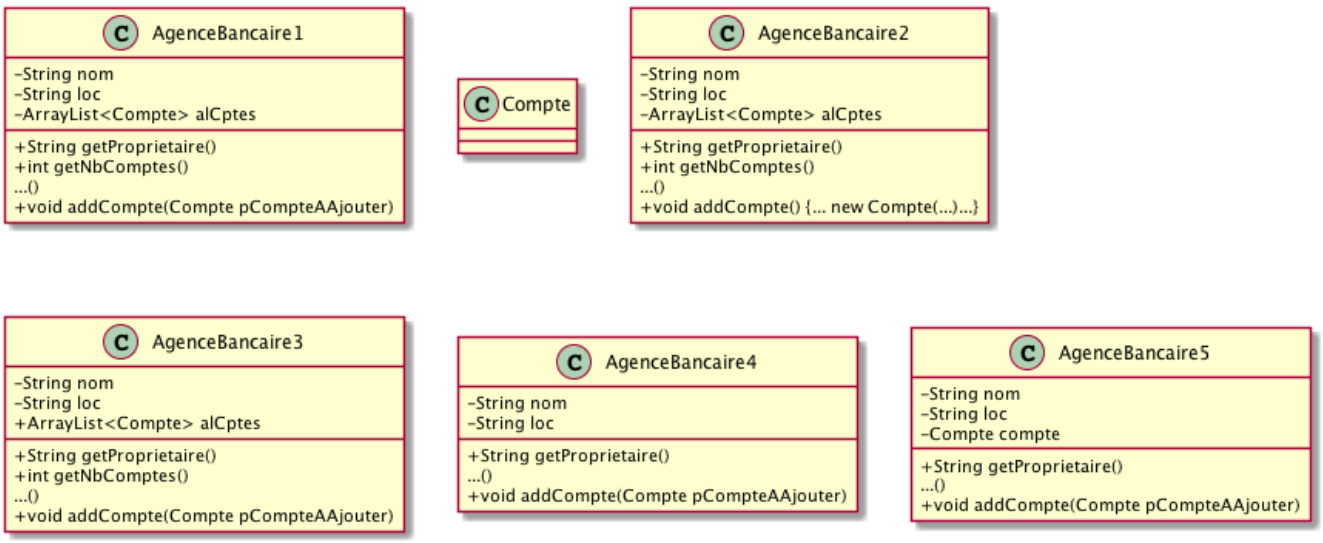

Figure 1. Diagramme de classe partiel

QUESTION

Complétez en ajoutant les relations (dépendance, association, composition, agrégation) entre les classes.

Solution

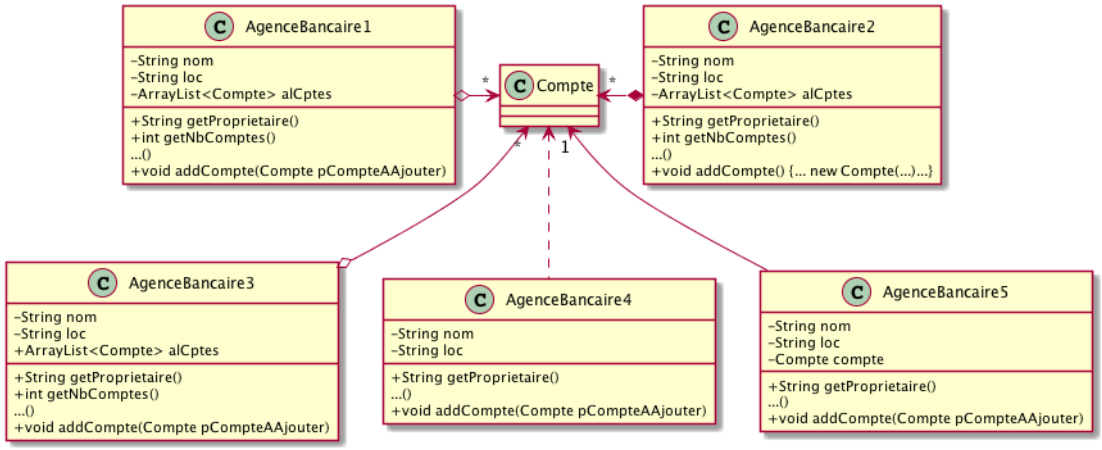

Figure 2. Diagramme de classe avec relations

2. Patrons

QUESTION

Pour chacun des diagrammes de classe partiels suivants (représentant des patrons que vous connaissez), complétez :

- le nom du patron dans la légende,
- en ajoutant les relations.

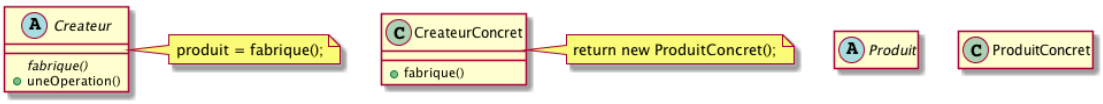

Figure 3. Patron ...

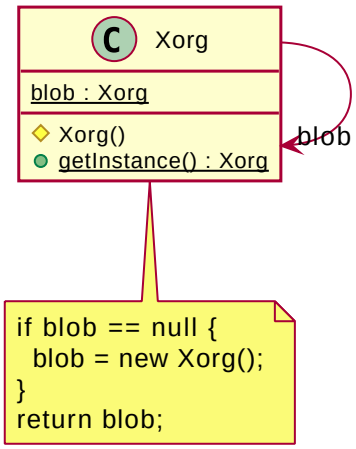

Figure 4. Patron ...

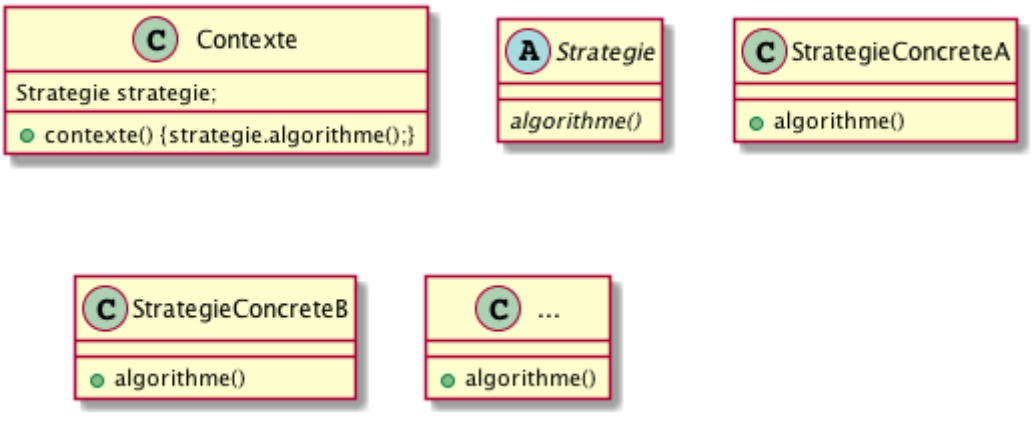

Figure 5. Patron ...

Solution

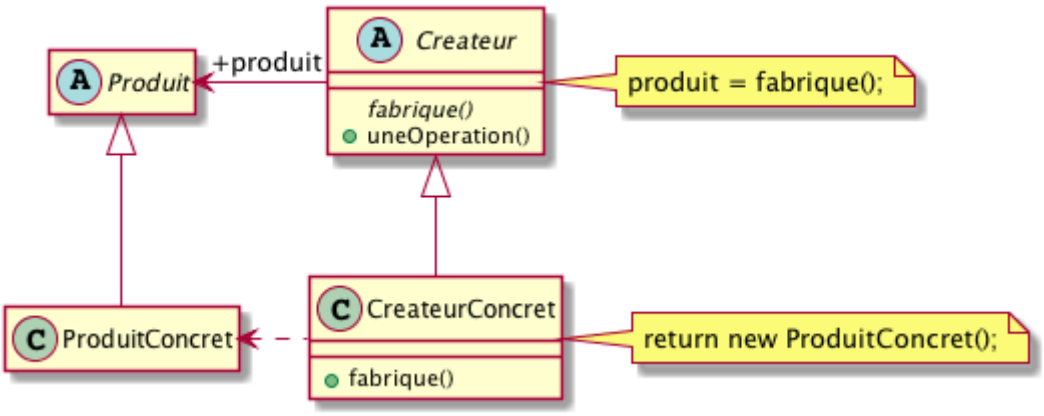

Figure 6. Patron Fabrique (simple)

Notez l'équivalence UML™ (ici noté en plantUML) entre :

- `Produit "+produit" <- Createur`
- `Produit "1" <- Createur {+Produit produit;}`

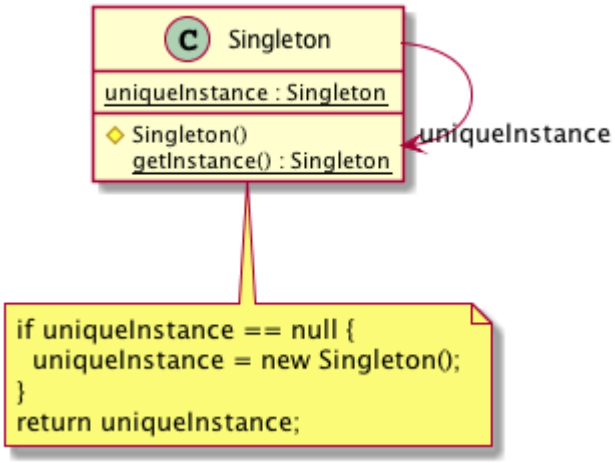

Figure 7. Patron Singleton

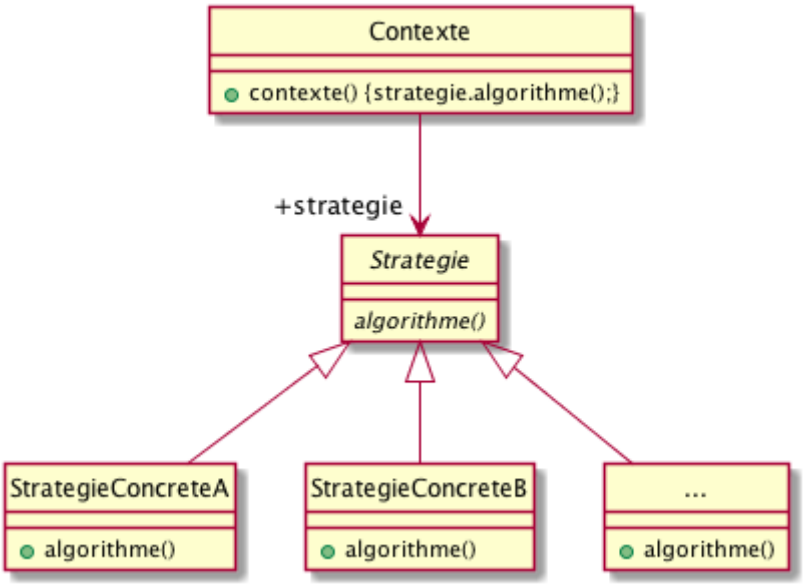

Figure 8. Patron Strategy

3. Diagrammes de séquences

Vous devez documenter, à partir des extraits de codes Java suivants, l'application `ApplicationBanque`, développée en S2.

Vous refactorerez cette application en TP, l'objectif n'est donc pas pour l'instant de remédier aux problèmes de conception mais plutôt de les identifier.

Méthode statique `comptesDUnPropretaire` *(*`ApplicationAgenceBancaire.java`*)*

```
public static void comptesDUnPropretaire (AgenceBancaire ag, String nomProprietaire) {
    Compte []  t;

    t = ag.getComptesDe(nomProprietaire);
    if (t.length == 0) {
        System.out.println("pas de compte à ce nom ...");
    } else {
        System.out.println("  " + t.length + " comptes pour " + nomProprietaire);
        for (int i=0; i<t.length; i++)
            t[i].afficher();
    }
}
```

QUESTION

Réalisez un diagramme de séquence illustrant le fonctionnement de cette méthode.

Solution 

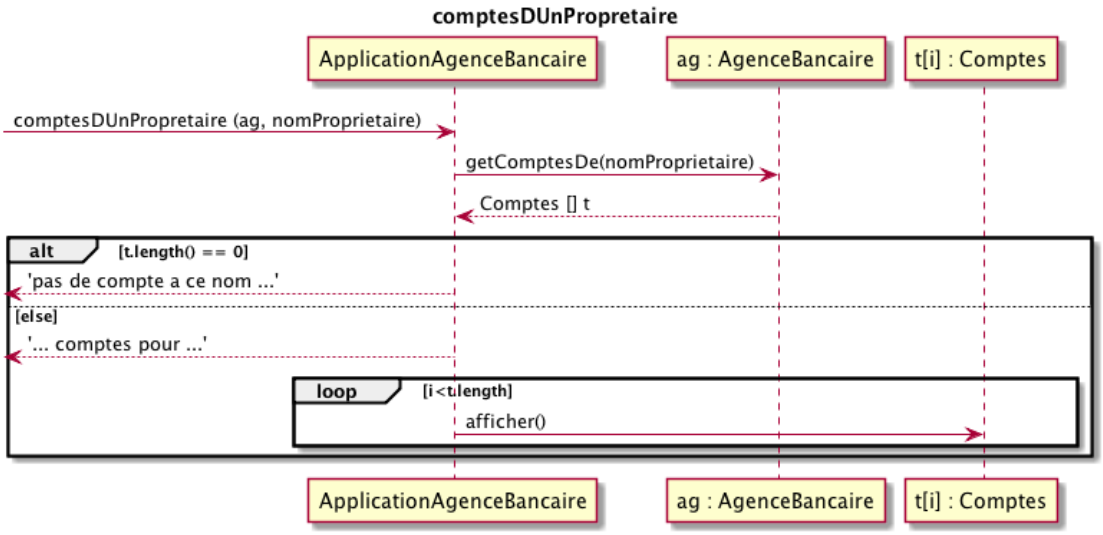

Figure 9. Diagramme de séquence de la méthode `comptesDUnPropretaire`

`ApplicationAgenceBancaire.java`

```
public class ApplicationAgenceBancaire {

    public static void main(String argv[]) {

        String choix;

        boolean continuer ;
        Scanner lect;
        AgenceBancaire monAg ;

        String nom, numero;
        Compte c;
        double montant;

        lect = new Scanner ( System.in );
        lect.useLocale(Locale.US);

        monAg = AccesAgenceBancaire.getAgenceBancaire();

        continuer = true;
        while (continuer) {
            ...
            choix = lect.next();
            choix = choix.toLowerCase();
            switch (choix) {
                case "q" :
                System.out.println("ByeBye");
                continuer = false;
                break;
                case "l" :
                monAg.afficher();
                break;
                case "v" :
                System.out.print("Num compte -> ");
                numero = lect.next();
                c = monAg.getCompte(numero);
                if (c==null) {
                    System.out.println("Compte inexistant ...");
                } else {
                    c.afficher();
                }
                break;
                case "p" :
                System.out.print("Propriétaire -> ");
                nom = lect.next();
                ApplicationAgenceBancaire.comptesDUnPropretaire (monAg, nom);
                break;
                case "d" :
                ...
                break;
                case "r" :
```

```
                ...
                break;
                default :
                ...
                break;
            }
        }
    }

    public static void comptesDUnPropretaire (AgenceBancaire ag,
        String nomProprietaire) {...}

    public static void deposerSurUnCompte (AgenceBancaire ag,
        String numeroCompte, double montant) {...}

    public static void retirerSurUnCompte (AgenceBancaire ag,
        String numeroCompte, double montant) {...}
}
```

Extrait de `AccesAgenceBancaire`

```
public class AccesAgenceBancaire {

    private AccesAgenceBancaire () {}
    public static AgenceBancaire getAgenceBancaire () {

        AgenceBancaire ag = new AgenceBancaire("CAISSE ECUREUIL", "PIBRAC");
        ...
    }
...
}
```

QUESTION

1. Réalisez le diagramme de classe de l'application
2. Que vous rappelle la classe `AccesAgenceBancaire`?
3. Réalisez un diagramme de séquence illustrant le fonctionnement de cette application (`main`). On utilisera des blocs "ref" pour les appels aux méthodes statiques, et on ne s'occupera pas des scanners.

Solution 

1. Diagramme de classe

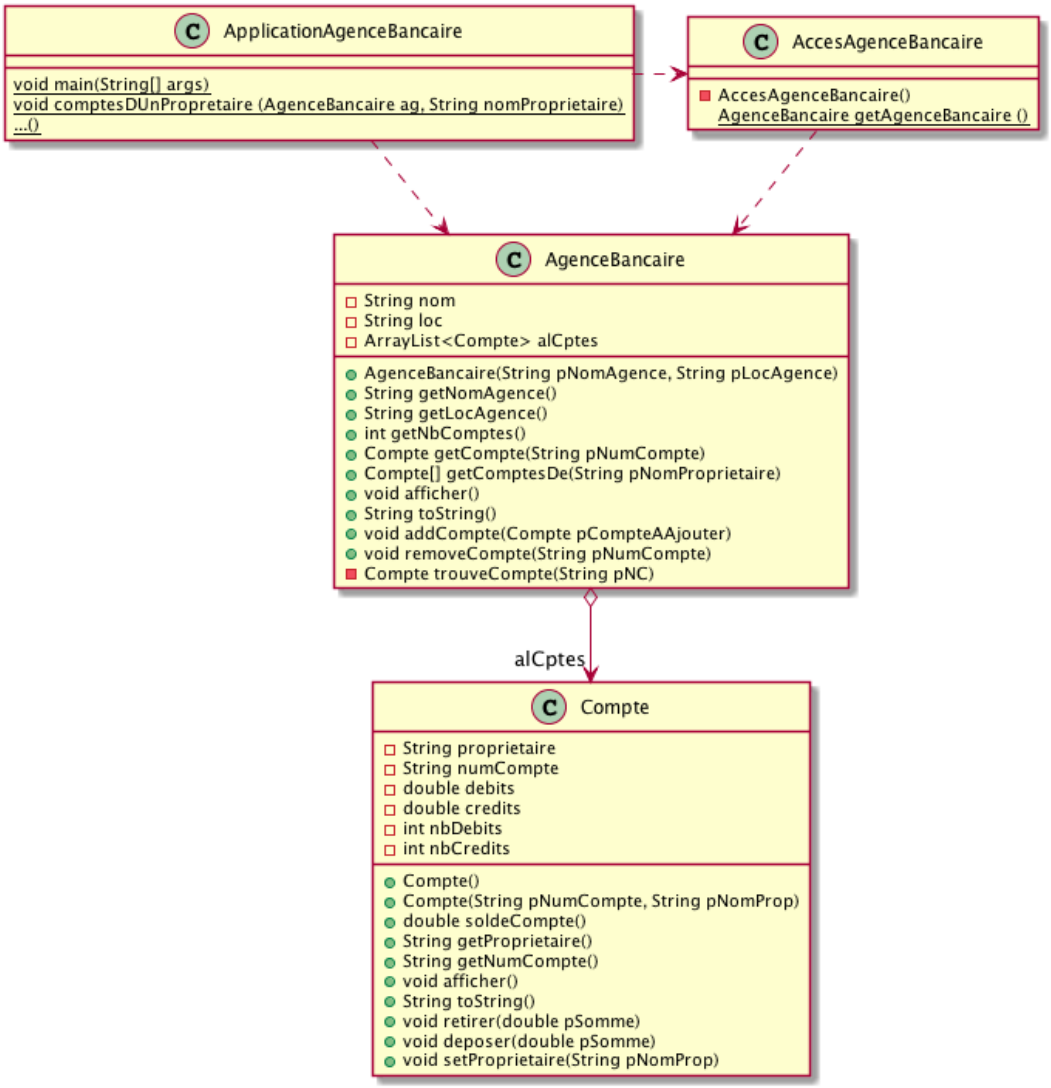

Figure 10. Diagramme de classe

Les étudiants ne peuvent avoir tous ces détails, mais ça leur servira pour le TP.

2. Cette s'approche de deux patrons que vous avez étudiés:
   - la structure du Singleton (getInstance() remplacé par getAgenceBancaire ()), sans la gestion de l'instance unique,
   - le comportement du créateur concrêt de la Factory.
3. Diagramme de séquence

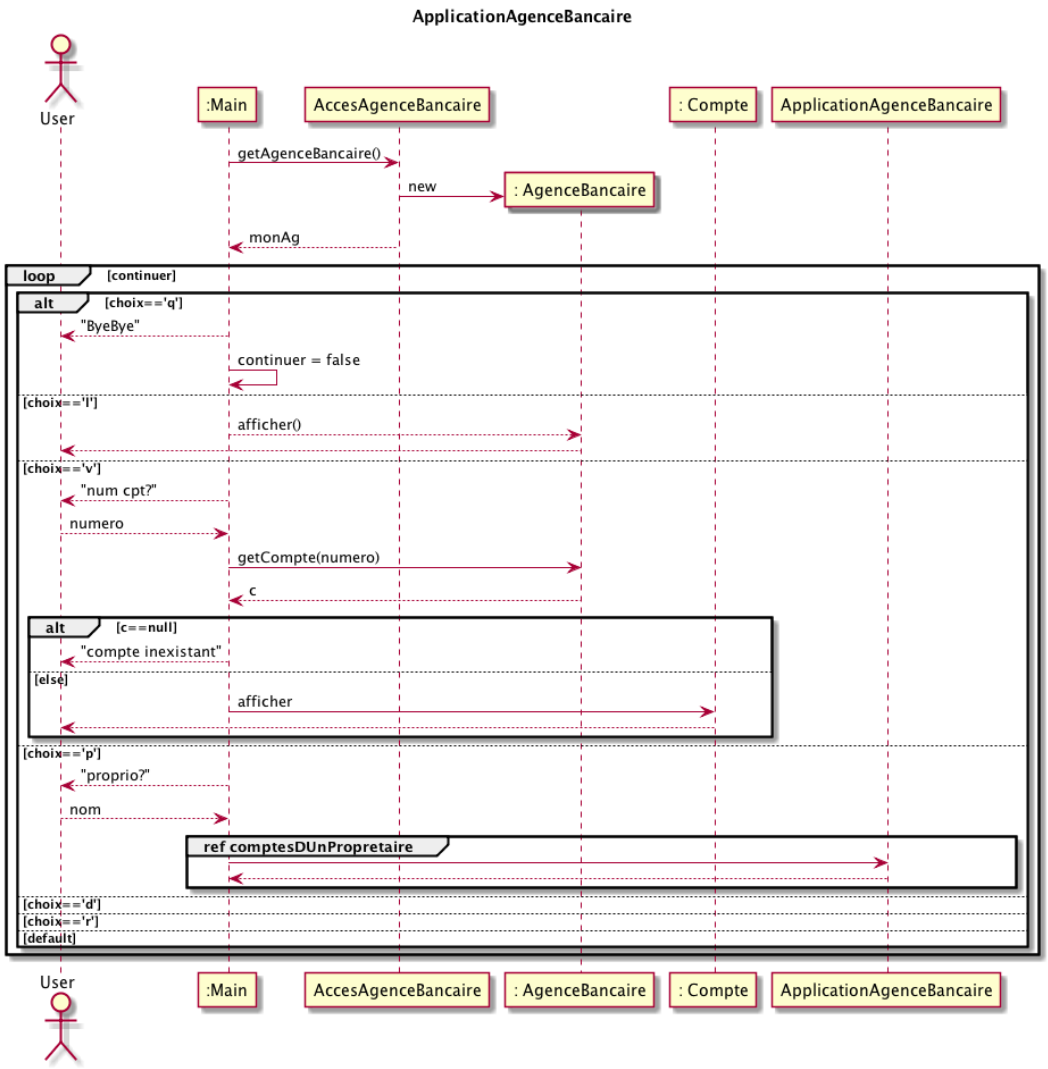

Figure 11. Diagramme de séquence de `ApplicationAgenceBancaire`

4. Machines à état

QUESTION

1. Dessinez le diagramme d'états correspondant aux feux tricolores (en France) classiques. Ajoutez la prise en compte de la panne dans un 2ème temps.
2. Dessinez le diagramme d'états correspondant au déroulement d'une partie d'échecs.

Solution 

1. Feu tricolore

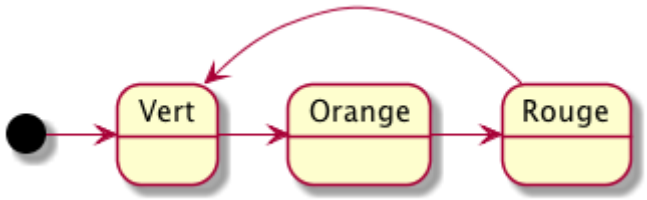

Figure 12. Diagramme d'état d'un feu tricolore classique

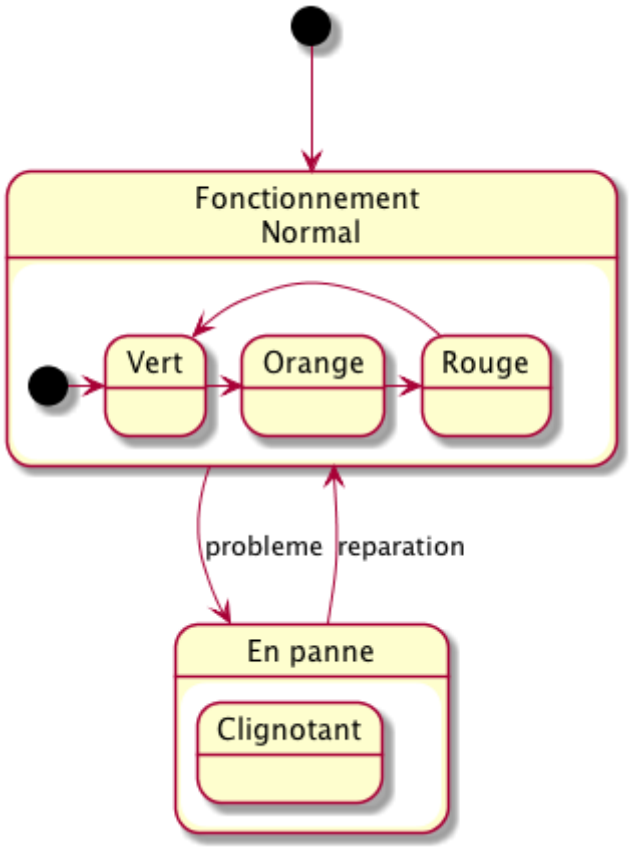

Figure 13. Diagramme d'état d'un feu tricolore avec panne

2. Echecs

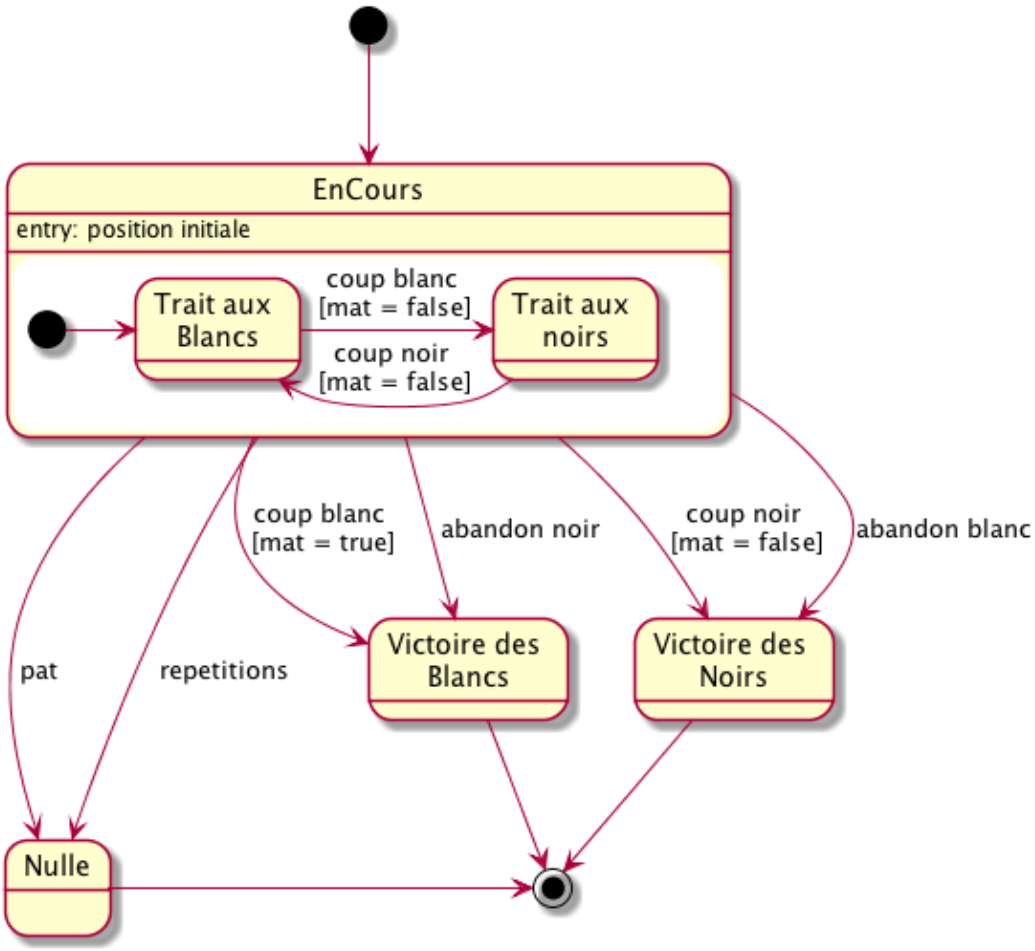

Figure 14. Diagramme d'état d'une partie d'échec

Pour aller plus loin

QUESTION

1. Peut-on, dans un code Java, faire la différence entre agrégation `1 <>-> *` et association `1 -> *`?

Solution 

1. En Java, un objet est toujours une référence, il n'y aura pas de changement de comportement de la JVM pour l'agrégation et la composition. Cependant, le modèle des classes et leur cycle de vie ne sera pas le même dans les deux cas:
    - Cas de la composition

```
public class Voiture {
    ArrayList<ComposantVoiture> mesParts;
    ...
    public Voiture (){
        mesParts.add(new ComposantVoiture("essieu"));
    }
    public void afficher () {
        mesParts.each...
            ...
            afficher();
    }
    public void detruire(){
        mesParts.each...
            ...
            detruire();
    }
}
```

    - Cas de l'agrégation

```
public class Voiture {
    ArrayList<Passager> passagers;
    public void ajouterPassager(Passager p) { passagers.add(p);
}
    public void detruire(){
        ...
    }
}
```