

Contrôle Long UML

Pierre Gérard

pierre.gerard@iutv.univ-paris13.fr

DUT Informatique S2D

Université de Paris 13

Résumé

Ce contrôle dure 3 heures. Aucun document n'est autorisé. Les durées ne sont données qu'à titre indicatif. Si vous êtes amenés à émettre des hypothèses, veuillez les expliciter sur la copie.

1 Questions de cours

Question : Suivant la terminologie employée dans la méthode utilisée dans le projet UML (Fée et Sorcière), décrivez les différents types de classes participantes.

Question : Que sont les Patrons de Conception (Design Patterns) et à quoi servent-ils ?

Question : Donnez un exemple de Patron de Conception (sauf Singleton) et utilisez un exemple pour détailler de quelle façon on l'utilise.

2 Diagramme de classes et OCL

Question : Etablissez le diagramme des classes correspondant au code source ci-dessous, aussi détaillé que possible. Utilisez une classe d'association lorsque c'est possible et pertinent.

```
class Personne{
    private String nom, prenom;
    public Personne getNom(){return nom;}
    public Personne getPrenom(){return prenom;}
    public void setNom(String nom){this.nom = nom;}
    public void setPrenom(String nom){this.prenom = prenom;}
    public Personne(String nom, String prenom){setNom(nom); setPrenom(prenom);}
    public boolean equals(Personne autre){
        return (getNom().equals(autre.getNom())
                && getPrenom().equals(autre.getPrenom()));
    }
}

class Musique{
    private Personne compositeur;
    public Personne getCompositeur(){return compositeur;}
    public void setCompositeur(Personne compositeur){this.compositeur = compositeur;}
    private String titre;
    public String getTitre(){return titre;}
    public void setTitre(String titre){this.titre = titre;}
    ...
}

class Chanson extends Musique{
    private Personne auteur;
    public Personne getAuteur(){return auteur;}
    public void setAuteur(Personne auteur){this.auteur = auteur;}
```

```

    ...
}

class Interpretation{
    private Chanson la_chanson;
    public Chanson getChanson(){return chanson;}
    public void setChanson(Chanson la_chanson){this.la_chanson = la_chanson;}
    private Personne interprete;
    public Personne getInterprete(){return interprete;}
    public void setInterprete(Personne interprete){this.interprete = interprete;}
    ...
}

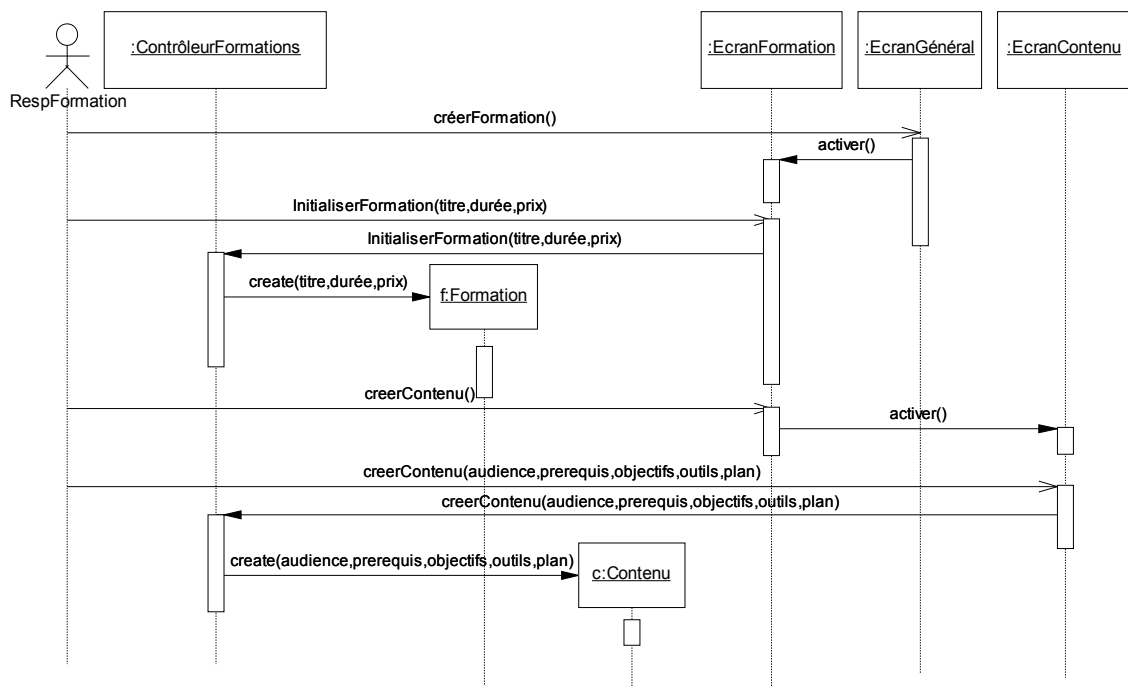
```

Question : Ecrivez des contraintes OCL pour spécifier les éléments suivants :

- Une chanson ne peut pas avoir d’auteur si elle n’a pas de compositeur.
- Le nom et le prénom d’une personne ne peuvent pas être des chaînes vides. On peut le faire avec une contrainte de type « inv » ou bien de type « pre ». Donnez les deux possibilités.

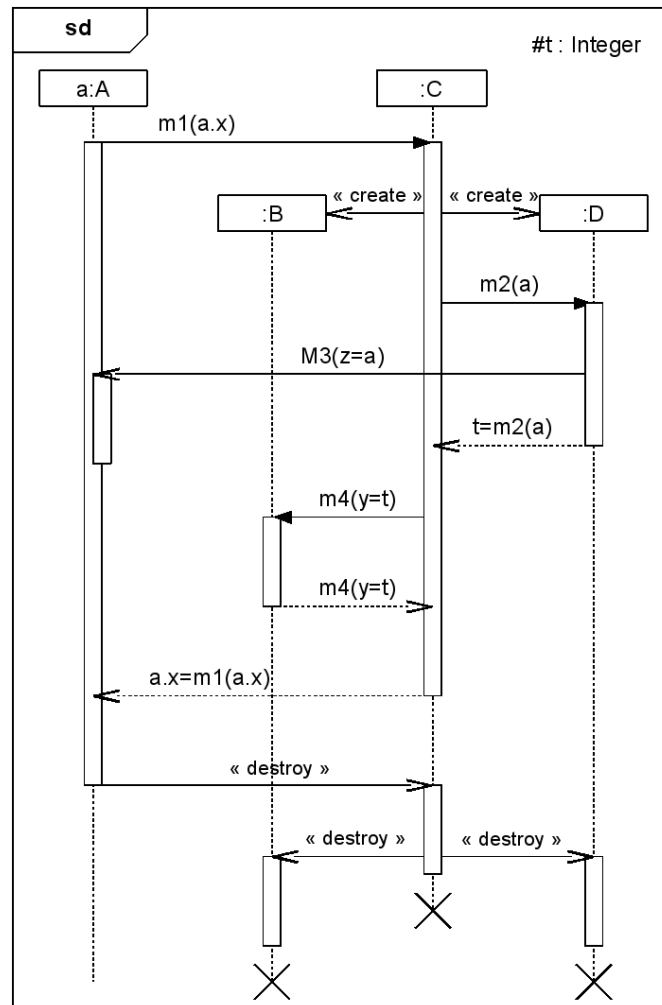
3 Diagrammes de séquences et de communication (2 pts)

Question : Transformez le diagramme suivant en un diagramme de communication équivalent.



4 Diagramme de classes et de séquences

Le diagramme de séquences ci-dessous fait appel à des éléments qui peuvent être définis dans un diagramme des classes. On supposera que a.x est de type Integer.



Question : Construisez un diagramme des classes cohérent avec ce diagramme de séquences. On veut un diagramme des classes simple mais qui définisse autant d'éléments que possible parmi ceux qui sont utilisés dans le diagramme de séquence.

5 Diagramme d'états (20 min)

Un Tamaguchi est un petit animal virtuel vendu sous la forme d'un objet électronique. L'utilisateur dispose d'un bouton pour « prendre soin » de cet animal virtuel.

Après avoir mangé, un Tamaguchi peut rester un certain temps dans son état normal sans être affamé. Ce temps est appelé temps d'autonomie. Ce laps de temps écoulé dans son état normal, le Tamaguchi a invariablement faim et dès qu'il a faim, il se met à pleurer. Pour lui donner à manger, l'utilisateur met le Tamaguchi à table. Dès qu'il est à table ce dernier cesse de pleurer et tant qu'il est en train de manger, il ne pleure pas. Passé un certain temps de restauration, il se remet à pleurer jusqu'à ce que l'utilisateur le sorte de table. A ce moment, le Tamaguchi revient dans son état normal... et ainsi de suite tant que le Tamaguchi ne meurt pas. Le Tamaguchi meurt s'il pleure plus de 5 minutes d'affilée.

Un Tamagushi est doté d'un bouton et il peut émettre des sons. Le bouton est utilisé par l'utilisateur pour mettre le Tamagushi à table où pour l'en sortir. D'un point de vue technique, une pression sur le bouton déclenche automatiquement l'opération `table()` du Tamagushi. Selon qu'il se mette à table ou qu'il en sorte, le Tamagushi génère des sons avec ses opérations `bipJaifini()` et `bipJemange()`.

Un Tamagushi est composé d'un afficheur et d'un générateur de nombres aléatoires. L'afficheur s'utilise au moyen de ses opérations `startPleurs()` et `stopPleurs()`. Le générateur est utilisé pour fournir des temps aléatoires d'autonomie et de restauration, au moyen de ses opérations `getTpsAutonomie()` et `getTpsRestauration()`.

Question : Proposez un diagramme de classes pour représenter un Tamagushi et ses composants

Question : Proposez un diagramme d'états associé à la classe Tamagushi qui permette de représenter le cycle de vie d'un Tamagushi. Utilisez des notations précises et en accord avec le diagramme de classes.

6 Diagramme d'activités

Considérons le flot d'activités du traitement d'une commande à la billetterie d'un théâtre. Après que le client ait sélectionné un type de place, on vérifie s'il dispose d'un abonnement. Dans le cas d'une commande isolée (hors abonnement), des sièges sont attribués puis la carte de crédit du client est débitée. Dans le cas où il s'agit d'une commande liée à un abonnement, en même temps que l'on attribue les sièges puis qu'on débite le compte du client, on applique une remise. Dans tous les cas, avant de finir le flot d'activités, on envoie les billets.

Question : Donnez le diagramme d'activités correspondant à la procédure décrite ci-dessus.