Le patron de Hiérarchie généralisée

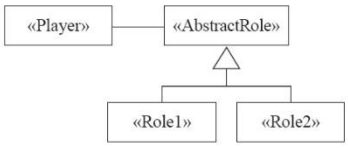
- ***Contexte***:
  - —Les objets d'une hiérarchie peuvent avoir un ou plusieurs objets au-dessus d'eux (leurs supérieurs),
    - Et un ou plusieurs objets sous eux (leurs subordonnés).
  - —Certains objets ne peuvent avoir de subordonnés
- ***Problème***:
  - —Comment représenter une telle hiérarchie d'objets dans laquelle certains objets ne peuvent avoir de subordonnés?
- ***Forces***:
  - —Vous recherchez une solution flexible afin de représenter une hiérarchie générale
  - —Les objets partagent plusieurs propriétés et comportements communs

Le patron Abstraction-Occurrence

- ***Contexte***:
  - —Souvent, dans le modèle du domaine, il existe un ensemble d'objets reliés entre eux (*occurrences).*
  - —Les membres d'un tel ensemble partagent des informations communes
    - Bien qu'ils diffèrent aussi.
- ***Problème***:
  - —Quelle est la meilleure façon de représenter un tel ensemble d'occurrences dans un diagramme de classes?
- ***Forces***:
  - —Vous souhaitez représenter les propriétés de chaque membre d'ensemble d'occurrences sans avoir à dupliquer l'information commune

Le patron Acteur-Rôle

- ***Contexte***:
  - —Un *rôle* correspond à un ensemble de propriétés particulières associées à un objet dans un contexte particulier.
  - —Un objet peut jouer plusieurs rôles dans différents contextes.
- ***Problème***:
  - —Comment modélisé une situation ou un objet peut jouer plusieurs rôles (conjointement ou consécutivement)?
- ***Forces***:
  - —Il est souhaitable de renforcer l'encapsulation en intégrant l'information associée à chaque rôle dans des classes distinctes.
  - —Il faut éviter l'héritage multiple.
  - —Un objet ne peut changer de classe d'appartenance
- ***Solution***:



Le patron Singleton

- ***Contexte***:
  - —Il est fréquent de retrouver des classes pour lesquelles il ne doit exister qu'une seule instance (*singleton)*
- ***Problème***:
  - —Comment assurer qu'il ne sera jamais possible de créer plus d'une instance de cette classe?
- ***Forces***:
  - —L'utilisation d'un constructeur public ne peut pas garantir qu'au plus une seule instance sera créée.
  - —L'instance du singleton doit être accessible de toutes les classes qui en ont besoin

The patron Observateur

- ***Contexte***:
  - —Quand une association est créée entre deux classes, celles-ci deviennent inséparables.
  - —Si vous souhaitez réutilisée l'une de ces classes, la seconde doit aussi être réutilisée.
- ***Problème***:
  - —Comment réduire l'interconnexion entre classes, en particulier si elle appartiennent à des modules ou des sous-systèmes différents?
- ***Forces***:
  - —Vous souhaitez maximiser la flexibilité du système

Les fragments permettent de décrire des diagrammes de séquence de manière compacte.

- **Alternative (alt)**, pour les alternatives avec conditions
- **Option (opt)**, pour un comportement optionnel
- **Boucle (loop)**, pour les boucles
- **Parallèle (par)**, pour un comportement concurrent

Le patron Délégation

- ***Contexte***:
  - —Vous devez concevoir une méthode dans une classe
  - —Vous réalisez qu'une autre classe possède une méthode qui fournit le service requis
  - —L'héritage n'est pas approprié
- ***Problème***:
  - —Comment réutiliser une méthode existant dans une autre classe?
- ***Forces***:
  - —Vous souhaitez minimiser le temps de développement par la réutilisation

Le patron Adaptateur

- ***Contexte***:
  - —Vous crécz une hiérarchie d'héritage et souhaitez y incorporer une classe existante (écrit par quelqu'un d'autre).
  - —Cette classe réutilisée fait aussi souvent déjà partie de sa propre hiérarchie d'héritage.
- ***Problème***:
  - —Comment bénéficier du polymorphisme en réutilisant une classe dont
    - les méthodes ont les même fonctions
    - mais *pas* la même signature

    que celles de la hiérarchie existante?
- ***Forces***:
  - —Vous n'avez pas accès à l'héritage multiple ou vous ne voulez

La patron Façade

- ***Contexte***:
  - —Souvent, une application contient plusieurs paquetages complexes.
  - —Un programmeur travaillant avec une librarie de classes doit manipuler de nombreuses classes
- ***Problème***:
  - —Comment simplifier la tâche des programmeurs lorsqu'ils interagissent avec une librairie complexe?
- ***Forces***:
  - —Il est difficile pour un programmeur de comprendre et d'utiliser un sous-système entier
  - —Si plusieurs classes d'une application appellent les méthodes de cette librairie, alors toute modification à celle-ci nécessitera une revue complète de tout le système.

Le patron Immuable

- ***Context***:
  - —Un objet immuable est un objet dont l'état ne change jamais
- ***Problème***:
  - —Comment créer un objet dont les instances sont immuables?
- ***Forces***:
  - —Il ne doit exister aucun moyen d'altérer l'état d'un objet immuable
- ***Solution***:
  - —S'assurer que le constructeur d'un objet immuable est le seul endroit ou les valeurs d'un objet sont fixées.
  - —Aucune méthode ne doit modifier l'état de l'objet.
  - —Si une méthode devrait avoir pour effet un changement d'état, alors une nouvelle instance est retournée.

Le patron en mode lecture seule

- ***Contexte***:
  - —Il faut parfois avoir certaines classes privilégiées ayant la capacité de modifier un objet immuable
- ***Problème***:
  - —Comment permettre une situation oèu une classe e en mode lecture seule pour certaines classes tout étant modifiable par d'autres?
- ***Forces***:
  - —Restreindre l'accès par les mot-clés **`public`**, **`protected`** et **`private`** n'est pas adéquat.
  - —Rendre **`public`** une méthode, la rend accessit tous

Le patron Mandataire

- ***Contexte***:
  - —Fréquemment, il est coûteux et complexe de créer une instance de certaines classes (ce sont des classes *lourdes*).
  - —Leur création exige du temps
- ***Problème***:
  - —Comment réduire la fréquence de création de classes lourdes?
- ***Forces***:
  - —En fonction des tâches à accomplir, tous les objets d'un système doivent demeurer disponibles lors de l'exécution du système.
  - —Il est aussi important d'avoir des objets dont les valeurs persistent d'une exécution à l'autre

Cohésion temporelle

Les opérations effectuées lors de la même phase d'exécution sont groupées ensemble

- Par exemple, tout le code utilisé lors de l'initialisation pourrait être regroupé.

Le patron Fabrique

- ***Contexte***:
  - —Un cadriciel réutilisable a besoin de créer des objets; toutefois les objets à créer dépendent de l'application.
- ***Problème***:
  - —Comment permettre à un programmeur d'ajouter des clases spécifiques à son application dans un système construit à parti d'un cadriciel?
- ***Forces***:
  - —Il faut que le cadriciel puisse créer des classes de l'application bien qu'il ne connait pas ces classes.
- ***Solution***:
  - —Le cadriciel délègue la création des classes à une classe spécialisée appelée la *Fabrique.*
  - —La fabrique est une interface générique définie dans le cadriciel.
  - —Cette interface contient une méthode dont le but est de créer des instance de sous-classes d'une classe générique.

Objectifs généraux d'un bonne design:

- Accroître les profits par la réduction des coûts et l'accroissement des revenus
- S'assurer de l'adhérence aux exigences
- Accélérer le développement
- Accroître les attributs de qualité tels
  - —Utilisabilité
  - —Efficacité
  - —Fiabilité
  - —Maintenabilité
  - —Réutilisabilité

hésion fonctionnelle

Lorsque que tout le code effectuant le calcul d'un certain résultat se trouve au même endroit

- i.e. lorsqu'un module effectue le calcul d'un seul résultat, sans effets secondaires.
- Bénéfices:
  - —Facilite la compréhension
  - —Plus facile à réutiliser
  - —Plus facile à remplacer
- Un module gérant une base de données, créant des fichiers ou interagissant avec un utilisateur n'est pas fonctionnellement cohésif

Cohésion en couches

Tous les fournisseurs d'accès à un ensemble de services interreliés sont groupés ensemble

- Les différentes couches devraient former une hiérarchie
  - —Les couches de plus haut niveaux peuvent accéder aux couches de plus bas niveaux,
  - —Les couches de bas niveaux n'accèdent pas aux couches de plus haut niveau
- L'ensemble des procédures qu'une couche met à la disposition des autres couches pour accéder aux services qu'elle offre est l'*application programming interface (API)*
- Une couche peut être remplacée sans que cela n'affecte les autres couche
  - —Il faut simplement reproduire le même API

Cohésion utilitaire

Les utilitaires interreliés sont rassemblés lorsqu'il n'y a aucun moyen de les regrouper en utilisant une forme de cohésion plus forte

- Un utilitaire est une procédure d'intérêt gé dans une grande variété d'applications.
- Les utilitaires sont hautement réutilisables
- Par exemple, la classe **`java.lang.Mat`**

Couplage de contenu

Lorsqu'une composante *subrepticement* modifie les données internes d'une autre composante

- Le fait d'encapsuler les données réduit considérablement le couplage
  - —Elles sont déclarées `private`
  - —Avec des méthodes `get` et `set`

Couplage d'estampillage

Lorsque les types d'arguments de méthode sont des primitives ou classes simples (par exemple, String)

- Plus il y a d'argument, plus ce couplage est fort
  - —Les méthodes appelantes doivent fournir tous ces arguments
- Il faut réduire ce type de couplage en évitant d'utiliser des arguments non-nécessaires
- Il y a souvent un compromis à faire entre couplage de données et couplage d'estampillage
  - —i.e. réduire l'un accroît l'autre

Lorsqu'une classe est déclarée dans la liste des arguments d'une méthode

- Une classe en utilise donc une autre
  - —Afin de réutiliser une classe, il faut aussi utilis l'autre
- Pour réduire ce type de couplage
  - —Utiliser une interface
  - —Transmettre que des variables simples

Cohésion séquentielle

Les procédures pour lesquelles l'une produit une sortie servant d'entrée à une autre sont groupées ensemble

- Ce genre de cohésion est valable lorsque les autres types de cohésion ont été achevés.

Cohésion procédurale

Les procédure qui se succèdent l'une après l'autre sont groupées ensemble

- Même si une ne produit pas un résultat utilisé par la suivante.
- Plus faible que la cohésion séquentielle.

Couplage de contrôle

Lorsqu'une procédure en appel une autre en utilisant une variable de contrôle ou une commande contrôlant l'exécution de la procédure appelée

- Afin d'effectuer un changement, il faut modifier à la fois l'appelé et l'appelant
- L'utilisation d'une opération polymorphique constitue la meilleure façon d'éviter le couplage de contrôle
- Une autre façon de réduire ce type de couplage consiste à avoir recours à une table *look-up*
  - —Chaque commande est alors associée une méthode qui sera appelée lorsque ces commande est lancée

Couplage d'appel

Lorsqu'une méthode en appelle une autre

- Ces méthodes sont couplée car le comportement de l'une dépend du comportement de l'autre
- Il y aura toujours du couplage d'appel dans tout système
- Si une même séquence d'appel se répète fréquemment, alors il faudrait songer à encapsuler cette séquence en créant une méthode regroupant ces appels

Couplage externe

Lorsqu'un module dépend d'une librairie, d'un système d'exploitation, d'un matériel

- Il faut réduire au maximum la dispersion de cette dépendance à travers le code.
- La Façade est un moyen efficace de réduire ce type de couplage

Couplage de type

Lorsqu'un module utilise un type défini dans un autre module

- Est présent chaque fois qu'une classe déclare un attribut d'une autre classe.
- La conséquence est que si le type (la classe) est modifié, alors la classe qui en fait usage devra aussi être changé
- Toujours utiliser le type le plus général

Couplage d'inclusion

Lorsqu'une composante en importe une autre

- (un paquetage en Java)

ou en inclut une autre

- (comme en C++).
- La composante qui procède à l'inclusion devient dépendante de la composante inclue.
- Si cette composante inclue est modifiée ou si quelque chose y est ajouté.
  - —Il peut se produire un conflit; un nouvel élément pouvant avoir le même nom qu'un élément existant.

Cohésion communicationnelle

Tous les modules accédant ou manipulant les même données sont groupées ensemble

- Une classe a une bonne cohésion communicationnelle si
  - —Toutes les opérations de manipulation de données sont contenus dans cette classe.
  - —La classe ne gère que les données qui la concerne.
- Avantage:
  - —Lorsqu'un changement doit être effectué sur les données, tout le code concerné se trouve au même endroit

Couplage commun

Lorsqu'une variable globale est utilisée

- Toutes les composantes utilisant cette variable globale deviennent alors couplées les unes aux autres
- Une forme plus faible de couplage est présente lorsque la variable est accessible à un nombre restreint de classes
  - —e.g. un paquetage Java
- Acceptable lorsque la variable globale fait référence à des paramètres globaux du système
- Le Singleton est un moyen d'offrir un accès contrôlé à un objet

Il vous est demandé de développer une application appelée HandyPerson qui permet aux clients d'embaucher un entrepreneur agréé pour leurs projets d'entretien, de réparation et de rénovation de la maison. Par conséquent, l'application supporte deux types d'utilisateurs : client et entrepreneur. Tous les utilisateurs sont identifiés par leur nom, leur adresse et leur e-mail.

Les entrepreneurs offrent des services en fonction de leur expertise (ex. finition des cloisons sèches, carrelage de plafond, réparation de robinets, etc.). Les services sont regroupés en catégories (ex. climat de la maison, réparations intérieures, rénovations extérieures, etc.) et sous-catégories (ex., systèmes de chauffage, moquette, patios, etc.).

Un client souhaitant souscrire au système doit fournir un numéro de carte de crédit (avec une date d'expiration). Les clients peuvent créer et soumettre des projets aux entrepreneurs via le système. Lorsqu'un client souhaite créer un projet, il navigue par catégorie pour voir les services disponibles. Chaque service listé est associé à un entrepreneur et a un taux horaire. Une fois que l'utilisateur sélectionne un service, les évaluations (reviews) et notation (rating) de l'entrepreneur qui fournit le service sont affichées. Le client spécifie ensuite la date de début du projet et le soumet. Ceci conclut le processus de création du projet.

L'entrepreneur peut voir la liste des projets provenant des clients. Il peut accepter ou rejeter chaque projet. S'il accepte le projet, il conclut un contrat contraignant avec le client.

Le client ou l'entrepreneur peut annuler le projet à moins qu'il ne soit prévu de commencer dans moins de 7 jours. L'entrepreneur enregistre le nombre d'heures de travail sur le projet via l'application. Une fois le projet est terminé, la carte de crédit du client est débitée. Cependant, l'argent n'est pas déposé dans le compte bancaire de l'entrepreneur tant que le client n'a pas indiqué sa satisfaction à l'égard du travail complété. Si le client n'est pas satisfait, un processus de médiation par un tiers peut entraîner un remboursement. Le processus de médiation dépasse le cadre de l'application.

Le paiement qui doit être effectué par le client est calculé en multipliant le nombre d'heures de travail effectué par l'entrepreneur par le taux horaire et en ajoutant un taux de transaction de 5%. Pour attirer des utilisateurs seniors, pour les clients âgés de 65 ans ou plus, le taux de transaction est réduit à 3%. Tous clients doivent être âgés de 18 à 125 ans.

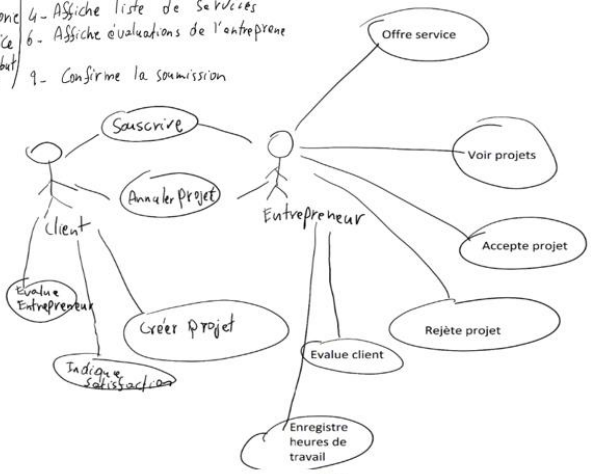
Un client qui a embauché un entrepreneur peut rédiger une évaluation sur la qualité de son travail et lui attribuer une notation. Les entrepreneurs peuvent également donner une notation aux clients avec lesquels ils ont travaillé, mais ne peuvent pas rédiger une évaluation.

1) Créez une liste de **cas d'utilisation** pour le système. Dessinez un **diagramme de cas d'utilisation** ET fournissez une description détaillée du cas d'utilisation « *Créer un projet* ».
2) Créez un **diagramme de classes** pour le système. N'incluez que les classes, associations, attributs et généralisations.
3) Dessinez un **diagramme d'état** pour la classe « Project ».
4) Dessinez un **diagramme de séquence** illustrant la séquence d'actions impliquées dans la création d'un projet.

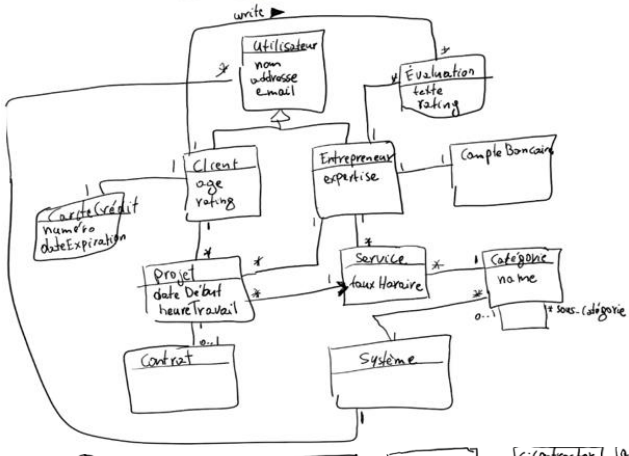
Client
- Souscrire
- Créer Projet
- Annule Projet
- Indique Satisfaction
- Evaluer Entrepreneur

Entrepreneur
- Offre Service
- Souscrire
- Voir Projets
- Accepte Projet
- Rejète Projet
- Annule Projet
- Enregistre heures de travail
- Evaluer client

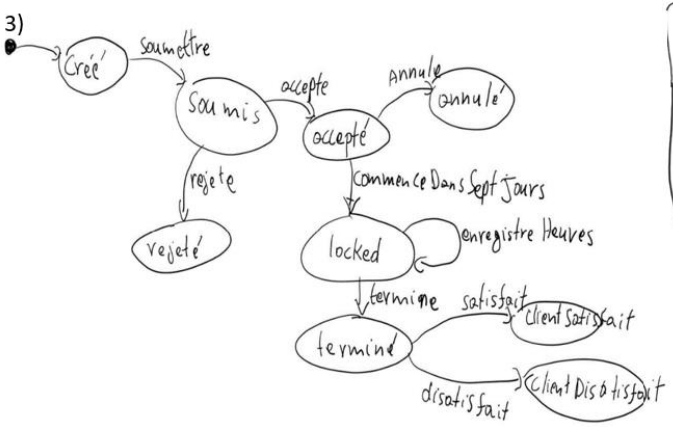
Etape Acteur	Réponses du Système
1- Demande la liste de catégories	2- Affiche liste de catégories
3- Selectionne catégorie	4- Affiche liste de Services
5- Selectionne service	6- Affiche évaluations de l'entreprene
7- Spécifie date de début	
8- Soumet le projet	9- Confirme la soumission



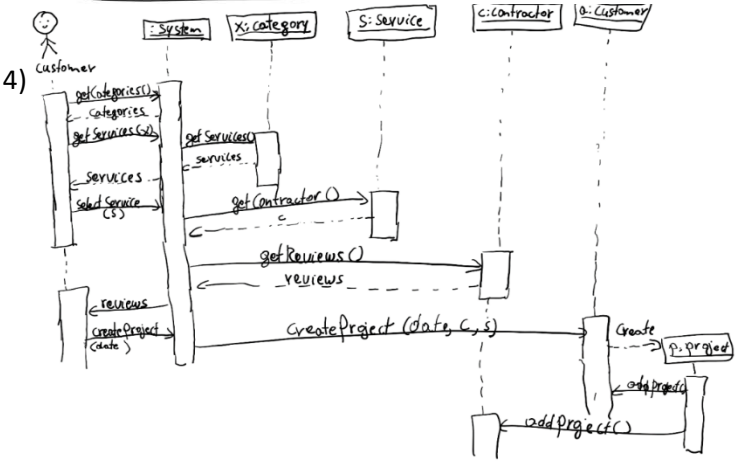
2)
- Système
- Utilisateur - nom, adresse, email
- Client - age, rating
- Entrepreneur - expertise, dateDebut, heuresDeTravail
- Projet
- Service - tauxHoraire
- Catégorie
- ~~Sous-catégorie~~
- CarteCrédit - numéro, dateExpiration
- Evaluation - texte, rating
- Contrat
- CompteBancaire



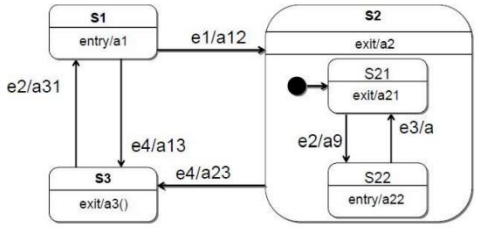
3)



4)



Question 11 - Consider the following UML state diagram. Assume you are already in state S1. What is the sequence of actions that will be executed If we input the following sequence of events: e1, e2, e4?



a. a12, a2, a21, a9, a22, a23
b. a12, a21, a9, a22, a2, a23
c. a12, a9, a21, a22, a23
d. a12, a21, a9, a22, a23, a3
e. None of the above