UE Conception Orientée Objet

Devoir Surveillé

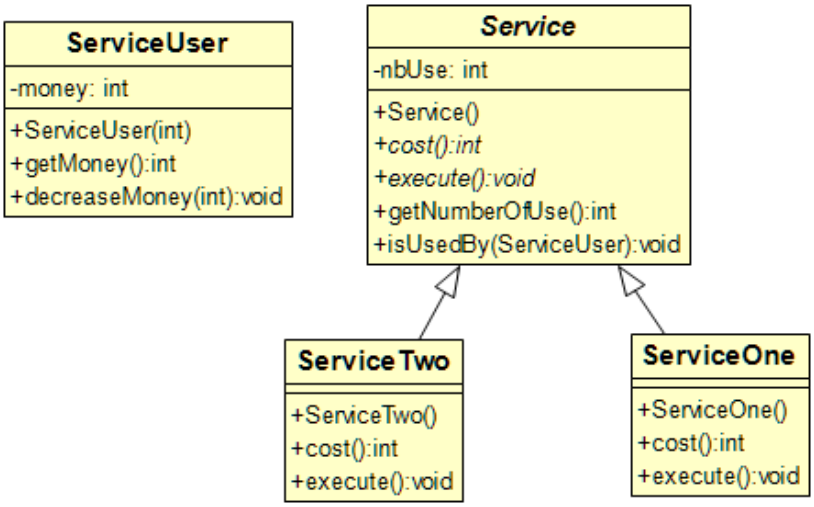
1 h 30

Copie des diapositives de cours annotée autorisée.
Dictionnaires de langue autorisés.
Autres documents interdits.

Indiquez sur votre copie *groupe INFO-X* ou *groupe MIAGE-X* où *X* est votre numéro de groupe.

Sauf mention expresse, la javadoc et les tests des classes à écrire ne sont pas demandés.

Exercice 1 :
On s'intéresse à des services qui peuvent être utilisés par des utilisateurs sous réserve que ceux-ci puissent les payer. Les classes qui apparaissent dans le diagramme suivant sont utilisées.



Le type `Service` permet de modéliser ces services. Le coût du service est fourni par la méthode `cost()` spécifique à chaque type de service. L'utilisation d'un service se fait par la méthode `isUsedBy`. Les noms des autres méthodes suffisent pour en comprendre les responsabilités.
Voici la documentation et la signature de `isUsedBy` :

```
/** The user uses this service. The user has to pay this service's cost.
 * @param user the user of this service
 * @throws NotEnoughMoneyException if the user has not enough money
 *                to pay this service's cost
 */
public final void isUsedBy(ServiceUser user) throws NotEnoughMoneyException
```

La spécification de `isUsedBy` précise qu'à l'issue de la méthode :

- le nombre d'utilisation du service a été incrémentée de 1 ;
- l'argent de l'utilisateur a été diminué du coût du service s'il en avait assez, sinon l'exception indiquée est levée.

Q 1 . Donnez le code des **classes de tests** (sauf leur entête et ses `import`) et leurs méthodes permettant de vérifier que l'implémentation de `isUsedBy` vérifie bien cette spécification quels que soient les services considérés, et en particulier pour les services de type « `ServiceOne` » ou ceux de type « `ServiceTwo` ».

Exercice 2 :
On considère des données météorologiques correspondant à des relevés réalisés par des stations météo. Ces données sont de type `WeatherData`. Elles sont caractérisées par la température et le volume des précipitations (*rainfall*) mesurés :

WeatherData
...
+ WeatherData(rainfall : float, temperature : float) + getRainfall() : float + getTemperature(): float

On s'intéresse en particulier à des traitements sur des listes de telles données. Par exemple on veut obtenir le cumul des précipitations ou la température minimale ou maximale sur ces données. D'autres traitements doivent pouvoir être envisagés. Ces traitements sont réalisés à l'aide du type `DataManager`.

Une première implémentation de `DataManager` est proposée via le code suivant (les packages sont omis) :

```
public enum Operation { totalRainfall, maxTemperature, minTemperature; }

public class DataManager {

  private Logger logger;
  private Operation operation;

  public DataManager(Operation op) {
    this.operation = op;
    this.logger = new Logger();
  }

  public float processData(List<WeatherData> datas) {
    this.logger.register(datas.size() + " data received");
    float result;
    // initialization
    switch (this.operation) {
      case totalRainfall:
        result = 0;
        break;
      case maxTemperature:
        result = -1000;
        break;
      default: //  == minTemperature
        result = +1000;
        break;
    }
    long start = System.currentTimeMillis();
    // process data
    for (WeatherData data : datas) {
      switch (this.operation) {
        case totalRainfall:
          result = result + data.getRainfall();
          break;
        case maxTemperature:
          result = Math.max(result, data.getTemperature());
          break;
        case minTemperature:
          result = Math.min(result, data.getTemperature());
          break;
      }
    }
    long end = System.currentTimeMillis();
    this.logger.register(" data processed in " + (end - start) + "ms");
    return result;
```

```
    }
}
```

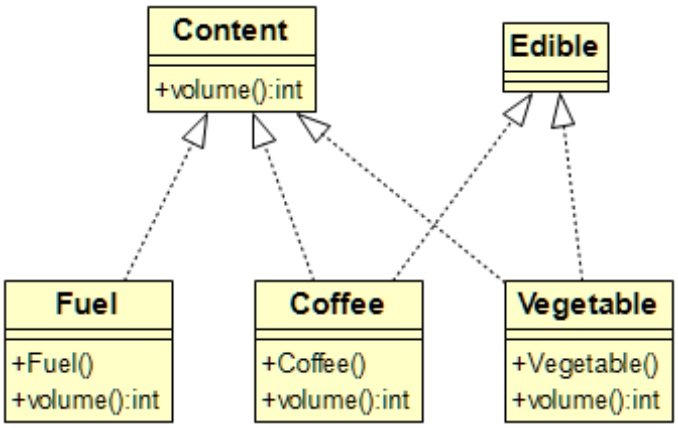
On suppose qu'est défini par ailleurs le type `Logger` dont la responsabilité et de permettre l'enregistrement de messages dans un fichier « de log » afin de garder une trace des différents calculs. Cet enregistrement est réalisé par la méthode `register`.

Q 1 . Indiquez concisément et précisément les défauts de ce code.

Q 2 . Sous la forme de diagramme UML, faites une proposition de conception qui corrige ce(s) problème(s).

Q 3 . Donnez le code java correspondant à votre classe qui permet de réaliser le traitement calculant la température maximale, **ainsi que** le code java de tous ses super-types (interfaces et/ou super-classes) (sauf `Object` bien sûr...).

Exercice 3 :
On s'intéresse ici à des contenus et aux récipients (`Container`) qui peuvent les contenir. Les contenus sont identifiés par l'implémentation de l'interface `Content` dont la méthode `volume()` permet de connaitre le volume représenté par le contenu.
Quelques contenus sont présentés dans le diagramme suivant. L'interface `Edible` permet de marquer les contenus alimentaires, et donc comestible (*edible*).



Les conteneurs sont spécialisés pour un type de contenu. Ce type de contenu est connu et fixé à la création du conteneur. On doit ainsi pouvoir créer

- des tasses à café, qui ne peuvent donc contenir que du café ;
- des boîtes de conserves, qui ne contiennent que des légumes ;
- des réservoirs d'essence qui ne contiennent que du carburant ;
- etc.

Un conteneur est caractérisé par le volume maximal de contenu qu'il peut accepter et, pour simplifier, le volume de son contenu est la seule information qu'il est nécessaire de gérer pour un conteneur.
Ces conteneurs doivent pouvoir invoquer les méthodes :

- `empty` qui vide ce conteneur de tout contenu ;
- `add` qui prend en paramètre un contenu à ajouter à ce conteneur.

  Cette méthode lève une exception `NotEnoughFreeSpaceException` si le volume du contenu ajouté est trop important pour le volume libre restant.
- `fillFrom` qui permet de remplir (éventuellement partiellement) ce conteneur à partir d'un autre conteneur *cont*, dont évidemment le contenu doit être compatible.

  Cette opération n'est réalisée que si il est possible de vider totalement *cont* dans ce conteneur, dans le cas contraire une exception `NotEnoughFreeSpaceException` est levée.

Q 1 . Donnez le code d'une classe `Container` correspondant à ce cahier des charges.

Vous donnerez la javadoc de votre méthode `add` (et uniquement celle là).

Q 2 . Donnez le code d'une classe `FuelTank` permettant de modéliser des réservoirs d'esssence.

Q 3 . Donnez le code d'une classe `ContainerForEdible` permettant de définir des conteneurs dont le contenu doit nécessairement être comestible.

Q 4 . Sans définir de nouvelle classe indiquer comment déclarer et initialiser une référence `mug` représentant un conteneur pour du café.

Quelles sont toutes les valeurs de type possible cette référence `mug`.