NFP121

Programmation avancée.

Session 1, 2 février 2021 - durée : 3 heures

Cnam / Paris-HTO & FOD/Nationale

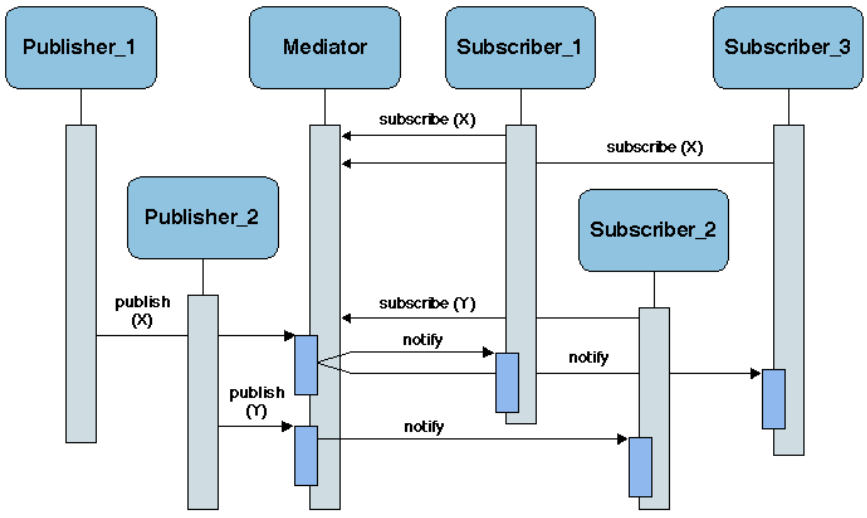
Sommaire :

Avertissement : Votre réponse à chaque question devra être soumise à l'outil JNEWS et votre travail devra être déposé avant 21h, toujours via Bluej item submit, **attention à l'échéance de 21h** l'outil n'a pas de tolérance...
Enfin, **toujours avant 21h** , générez l'archive java de votre travail en **préfixant avec votre NOM**, par ex. **MARTIN_ExamenFevrier2021.jar** (sous Bluej menu Projet item Create Jar File/Exporter)
Puis

- déposez cette archive sur votre agenda, comme un tp ordinaire (sans rapport cette fois)
- **et avant 21h15** envoyez moi cette même archive (ex: **MARTIN_ExamenFevrier2021.jar)** via WeTransfer (https://wetransfer.com/) à jean-michel.douin@lecnam.net, en indiquant vos nom/prénom/identifiant_Cnam, vous recevrez ainsi un accusé de la réception de votre travail.

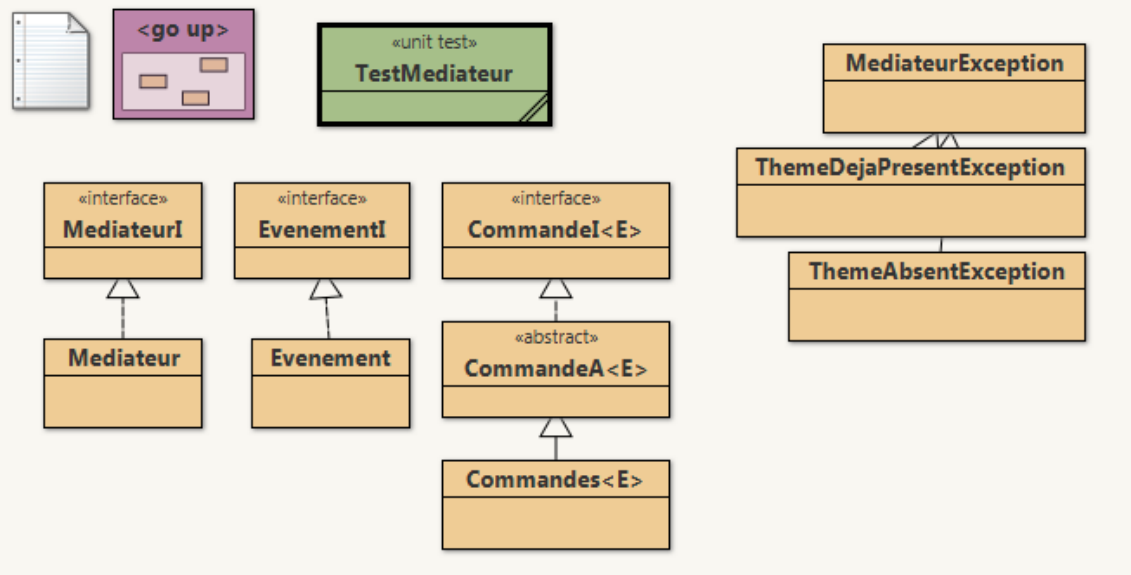
Question1 : Le Patron *PublishSubscribe*

L'utilisation de ce patron permet à un souscripteur à un thème donné, d'être notifié à la publication d'un message sur ce thème. L'enregistrement des souscripteurs se fait auprès d'un médiateur qui se charge de la publication.



source: http://lig-membres.imag.fr/krakowia/Files/MW-Book/Chapters/Events/events-body.html

Ci-dessous, en syntaxe UML/BlueJ, une architecture des classes:



Cette architecture est constituée des interfaces ***MediateurI***, ***EvenementI***. A ces interfaces correspondent les implémentations ***Mediateur*** et ***Evenement***. Le médiateur se charge de la publication d'un évènement sur un thème donné en déclenchant la méthode ***executer*** issue de l'interface ***CommandeI*** (cf. le patron Commande). L'utilisation de ce patron permet de s'affranchir de futures implémentations de souscripteurs, (i.e. à la question 2 les souscripteurs utiliseront le patron décorateur et le patron chaîne de responsabilités à la question 3).

L'interface ***MediateurI*** propose les opérations :

- d'ajout d'un thème de publication (***`ajouterTheme`***),
- de publication d'un évènement (**`publier`**),
- de retrait d'un thème de publication (**`retirer`**),
- d'obtention d'un itérateur sur les thèmes ajoutés (**`iterator`**),
  note: La méthode remove de l'itérateur retourné est sans effet
- d'une définition de l'invariant de classe (repOk),
- d'une fonction d'abstraction (af) qui est **optionnelle**.

```
public interface MediateurI extends Iterable<String>{

    /** Obtention du nom du médiateur.
     * @return le nom de médiateur, (ce nom ne peut-être == null)
     */
    public String getNom();

    /** Ajout d'un thème de publication.
     * @param theme le thème à ajouter
     * @param commande la commande exécutée lors d'une publication sur ce thème
     * @throws ThemeDejaPresentException si le thème est déjà présent
     */
    public void ajouterTheme(String theme, CommandeI<EvenementI> commande) throws
ThemeDejaPresentException;

    /** Retrait d'un thème de ce médiateur.
     * @param theme le thème à retirer
     * @throws ThemeAbsentException si le thème est absent du médiateur
     */
    public void retirerTheme(String theme) throws ThemeAbsentException;

    /** Publication d'un évènement avec ce thème.
     * @param evenement l'evénement contient le thème de publication
     * @throws ThemeAbsentException si le thème est absent du médiateur
     */
    public void publier(EvenementI evenement) throws ThemeAbsentException;
```

```
    /** Itérateur du médiateur.
     * la méthode remove de l'itérateur obtenu est sans effet
     * @return un itérateur sur les tous les thèmes présents
     */
    public Iterator<String> iterator();

    /** Invariant de classe.
     * @return vrai si l'invariant est satisfait
     */
    public boolean repOk();

    /** Fonction d'abstraction. optionnelle, optionnelle
     * @return une représentation dite abstraite
     */
    public <T> T af();
}
```

L'interface ***EvenementI***, suivie de ***CommandeI*** :

L'interface ***EvenementI*** propose les opérations :

- d'affectation du thème de publication pour cet évènement (**`setTheme`**),
- d'ajout de paramètres à transmettre (**`putParametre`**),
- d'obtention des paramètres associés à cet événement(**`getParametre, getParametres)`**
- d'affectation et de lecture de la priorité, etc..
- La relation d'ordre est effectuée selon la priorité de l'événement
  Valeur élevée implique une priorité forte
- d'une définition de l'invariant de classe (repOk),
- d'une fonction d'abstraction (af) **optionnelle**.

La classe ***Evenement*** est fournie et implémente les méthodes de l'interface, seule la relation d'ordre entre événements (cf. interface Comparable<***EvenementI***>) est à développer. Les classes des exceptions sont complètes.

```
public interface EvenementI extends Comparable<EvenementI>{

  public void setTheme(String theme);

  public String getTheme();

  public void putParametre(String clef, Object valeur);

  public <T> T getParametre(String clef);

  public Map<String,Object> getParametres();

  public String toString();

  public void setPriorite(int priorite);

  public int getPriorite();

    /** Invariant de classe.
     * @return vrai si l'invariant est satisfait
     */
    public boolean repOk();

    /** Fonction d'abstraction. optionnelle, optionnelle
     * @return une représentation dite abstraite
     */
    public <T> T af();
}
```

L'interface ***CommandeI*** propose l'opération générique **executer** effectuée à chaque publication

```
/**
 * @param <E> l'entité transmise
 */
public interface CommandeI<E>{


  /* La méthode exécutée lors d'une publication.
   * @param e l'entité, l'événement transmis
   * @return true ou false en fonction du contexte, et du résultat attendu de l'exécution
   */
  public boolean executer(E e);

  public final static int PRIORITE_PAR_DEFAUT = 5;

  /* Accesseur/mutateur pour la priorité d'une commande
   */
  public int getPriorite();
  public void setPriorite(int priorite);

  /** Accesseur pour le nom de la commande */
  public String getNom();


}
```

Question1-1)
Proposez une implémentation **complète de la classe *Mediateur***.

Question1-2)
Ecrivez une implémentation **complète de la classe *Commandes.*** Cette classe permet d'exécuter une liste de commandes en séquence, complétez également la relation d'ordre entre événement cf. la classe ***Evenement***.

Question1-3)
Un tri peut être effectuée sur cette liste de commandes, la publication s'effectuera alors selon cet ordre. La relation d'ordre pour les commandes est transmise lors de l'appel
de la méthode ***setComparator(Comparator<CommandeI<E>> comparator)***,

de quel patron s'agit-il ? notez le en commentaires de cette méthode.

- **La classe de tests unitaires fournie doit être lue attentivement et ne doit pas être modifiée et s'exécuter sans erreur.**

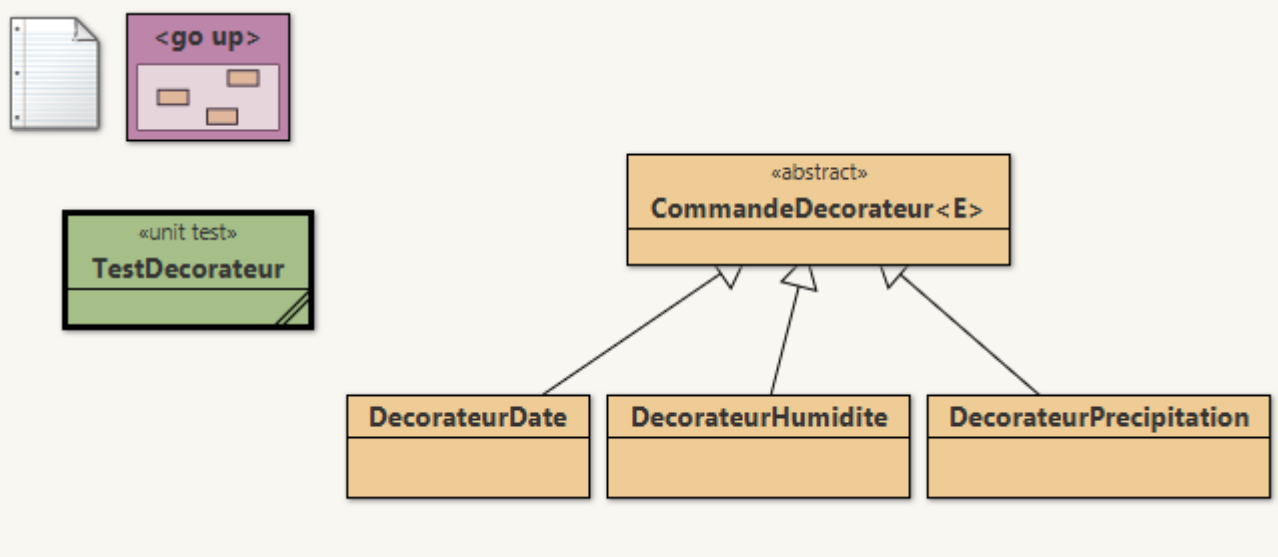
- **N'oubliez pas de soumettre cette question à l'outil JNEWS**
- N'oubliez pas de mettre votre nom au début de chaque classe que vous avez complétée
- dans les commentaires au début de chaque classe /** @author MARTIN .... */

Question2 : Patron Décorateur

La commande exécutée à la suite d'une publication déclenche une instance patron décorateur. Appliqué à l'exemple de la météo, chaque décorateur se charge de l'affichage de l'un des champs issu de l'évènement reçu (soit la date, le taux d'humidité, les précipitations, ...).

L'exécution d'un décorateur se contente d'afficher sur la console la valeur du champ choisi (Decorateur**Date**, Decorateur**Humidite**, Decorateur**Precipitation**).

Une version "lisible" de la commande ainsi décorée est réalisée par l'appel de la méthode ***toString***.



Attention le décorateur chargé de l'humidité ne "propage" pas l'évènement reçu si la valeur de l'humidité est supérieure à 75, cf. la classe de tests unitaires **TestDecorateur**

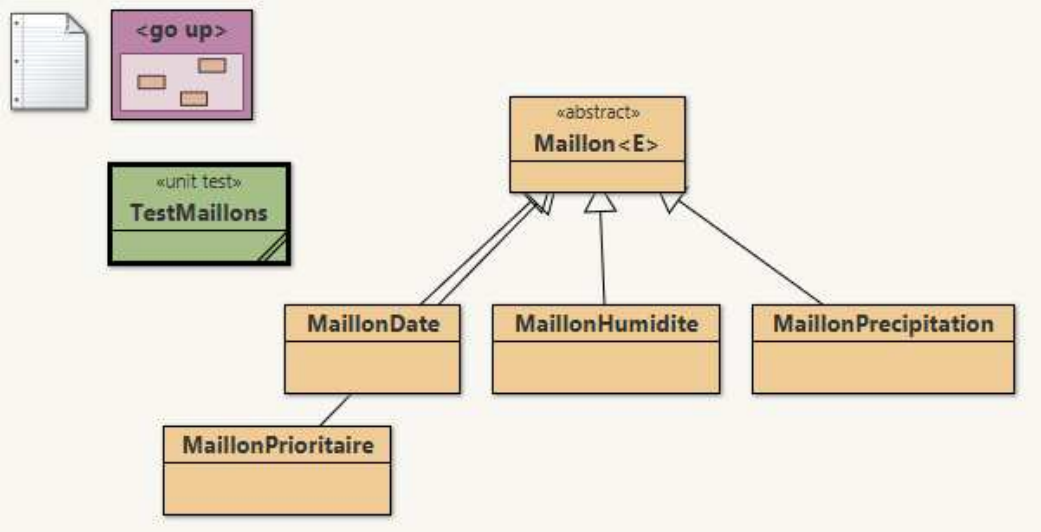
Question2)
Complétez toutes les classes issues du patron décorateur.

- **La classe de tests unitaires fournie doit être lue attentivement et ne doit pas être modifiée et s'exécuter sans erreur.**
- **N'oubliez pas de soumettre cette question à l'outil JNEWS**
- N'oubliez pas de mettre votre nom au début de chaque classe que vous avez complétée
  - dans les commentaires au début de chaque classe /** @author MARTIN .... */

Question3 : Le patron Chaîne de responsabilités

La commande exécutée à la suite d'une publication déclenche une instance du patron **Chaîne de responsabilités**, toujours appliqué à l'exemple de la météo. Chaque maillon de la chaîne se charge de l'affichage de l'un des champs issu de l'évènement reçu (Maillon**Date**, Maillon**Humidite**, Maillon**Precipitation**).

L'exécution se contente simplement d'afficher sur la console la valeur du champ choisi, une version lisible par l'appel de la méthode ***toString*** est réalisée en fonction des maillons exécutées de la chaîne des mailons.



Attention

- le maillon chargé de l'humidité ne transmet pas l'évènement reçu si le taux d'humidité est supérieur à 75, cf. la classe de tests unitaires
- le maillon ***MaillonPrioritaire*** ne transmet pas l'évènement si la priorité de l'événement est supérieure à une borne transmise à la création d'une instance de ce maillon.

Question3-1)
Complétez toutes les classes issues du patron Chaîne de responsabilités.

Question3-2)
Complétez la classe **Introspection**, qui contient une méthode classe permettant d'ajouter un successeur à l'un des maillons de la chaîne, le nom du successeur est en paramètre, cf. la méthode **testParIntrospection** de la classe de tests unitaires.

Question3-3)
Quelles sont, quelle est la différence entre le patron Décorateur et le patron Chaîne de responsabilités notez votre réponse dans la classe de tests en commentaires Java.

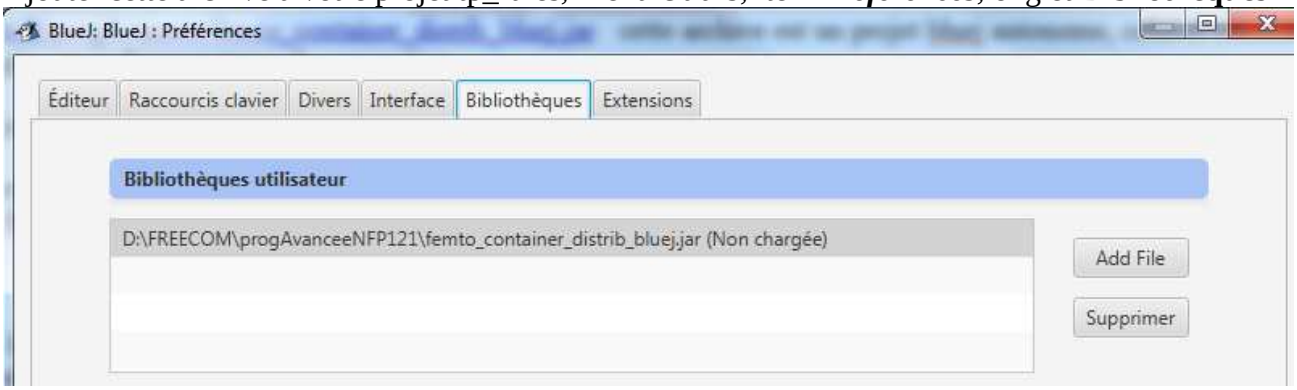
- **La classe de tests unitaires fournie doit être lue attentivement et ne doit pas être modifiée et s'exécuter sans erreur.**
- **N'oubliez pas de soumettre cette question à l'outil JNEWS**
- N'oubliez pas de mettre votre nom au début de chaque classe que vous avez complétée
  - dans les commentaires au début de chaque classe /** @author MARTIN .... */

Question4: Décorateurs injectés et l**e patron *ServiceLocator***

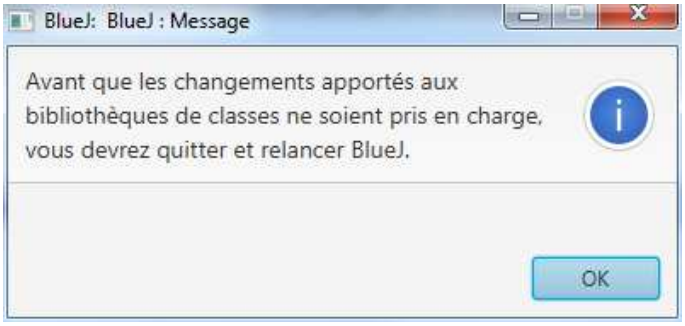
Question4-1 : Les décorateurs sont injectés

***Préambule** BlueJ : **import de la librairie** **femtoContainer** (si ce n'est déjà fait cf. tp_rules,tp_injection...)*

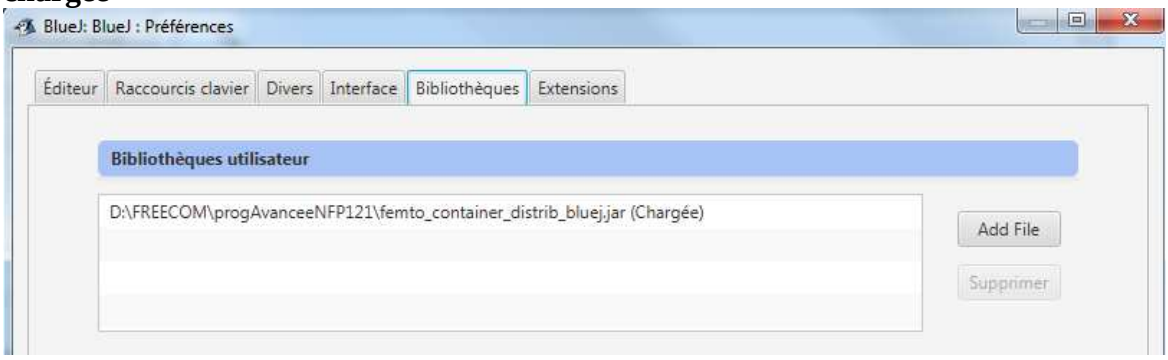
1. Téléchargez femtoContainer.jar
2. Ajouter cette archive à votre projet tp_rules, Menu ***Outils***, item ***Préférences***, onglet **Bibliothèques**



3. Redémarrez BlueJ



4. Vérifiez à nouveau Menu ***Outils***, item ***Préférences***, onglet **Bibliothèques**, l'archive doit être **chargée**



Des instances issues du patron Décorateur utilisées en question2 sont maintenant injectées via l'outil **femtoContainer** vu en cours et utilisé en TP.

Question4-1)

Ajoutez le décorateur de votre choix, modifiez le fichier de configuration question4/config.txt, enrichissez la classes de Tests unitaires **sans et avec** l'usage de ***femtoContainer***.

Commentez en une ou deux phrases cette approche nommée injection de dépendances/Inversion de contrôles, avantages, inconvénients... Mettez votre réponse dans la classe de tests unitaires en commentaires pour Java.

Le fichier de configuration pour l'outil **femtoContainer** se trouve dans le répertoire **question4/config.txt**, soit :

 # retirez le caractère # pour votre réponse en Question4-1
#bean.id.6=decorateurXXXXX
 # le nom du décorateur créé question4.DecorateurXXXXX
decorateurXXXXX.class=question4.DecorateurXXXXX
decorateurXXXXX.property.1=commande
decorateurXXXXX.property.1.param.1=decorateurYYYYY

bean.id.1=mediateur
mediateur.class=question1.Mediateur
mediateur.property.1=nom
mediateur.property.1.param.1=mediateur1

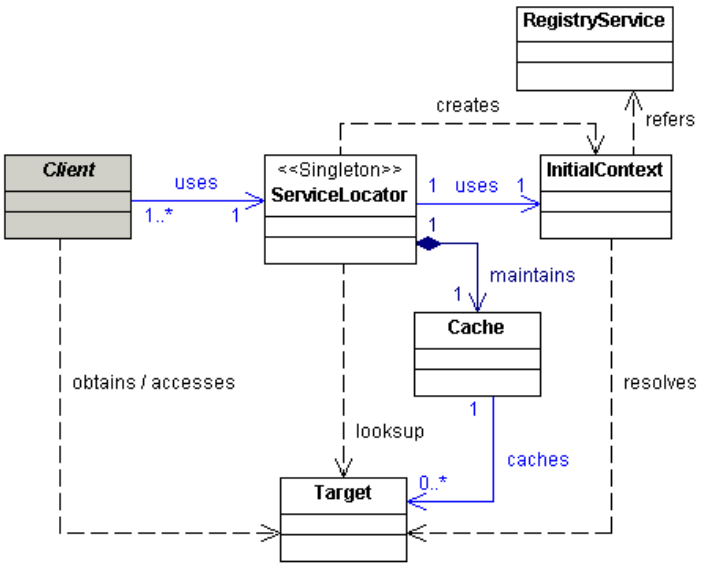
#testCommande = new TestCommande("commandeMeteo");
bean.id.2=testCommande
testCommande.class=question4.TestDecorateur$TestCommande
testCommande.property.1=nom
testCommande.property.1.param.1=commandeMeteo

#
bean.id.3=decorateurPrecipitation
decorateurPrecipitation.class=question2.DecorateurPrecipitation
decorateurPrecipitation.property.1=commande
decorateurPrecipitation.property.1.param.1=testCommande

bean.id.4=decorateurHumidite
decorateurHumidite.class=question2.DecorateurHumidite
decorateurHumidite.property.1=commande
decorateurHumidite.property.1.param.1=decorateurPrecipitation

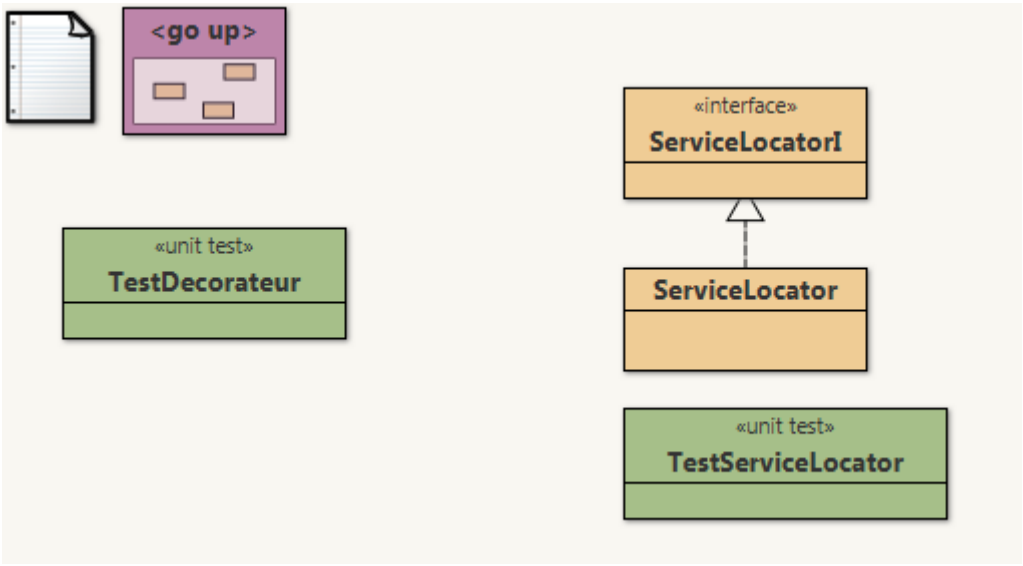
bean.id.5=chaineDeCommandes
chaineDeCommandes.class=question2.DecorateurDate
chaineDeCommandes.property.1=commande
chaineDeCommandes.property.1.param.1=decorateurHumidite

Question4-suite: Le patron ServiceLocator



source: http://www.corej2eepatterns.com/ServiceLocator.htm "**Use a Service Locator to implement and encapsulate service and component lookup. A Service Locator hides the implementation details of the lookup mechanism and encapsulates related dependencies.**"

Une instance de la classe ***ServiceLocator*** contient plusieurs médiateurs et autorise une recherche des différents services qui les composent. La configuration est laissée à la responsabilité de l'outil ***femtoContainer***, plusieurs médiateurs peuvent y être installés et référencés.



Question4-2)

Proposez une implémentation du patron ServiceLocator, (les recherches de type annuaire du schéma i.e les classes RegistryService et InitialContext doivent être ignorées).

Ajoutez le médiateur de votre choix, un évènement de votre choix, enrichissez la classes de Tests unitaires **sans et avec** l'usage de ***femtoContainer***.

Que doit-on satisfaire afin qu'une instance d'un ***ServiceLocator*** devienne à son tour configurable via ***femtoContainer ?***

Mettez votre réponse dans la classe de tests unitaires en commentaires pour Java.

Le fichier de configuration pour l'outil femtoContainer se trouve dans le répertoire **question4/README.txt**, l'icône document en haut à gauche de votre page Bluej attachée au répertoire question4 soit :

```
# mediateur_1 = new Mediateur("mediateur1");
bean.id.1=mediateur_1
mediateur_1.class=question1.Mediateur
mediateur_1.property.1=nom
mediateur_1.property.1.param.1=mediateur1

# mediateur_2 = new Mediateur("mediateur2");
bean.id.2=mediateur_2
mediateur_2.class=question1.Mediateur
mediateur_2.property.1=nom
mediateur_2.property.1.param.1=mediateur2

#testCommande = new TestCommande("commandeMeteo1");
bean.id.3=testCommande
testCommande.class=question4.Z_TestServiceLocator$TestCommande
testCommande.property.1=nom
testCommande.property.1.param.1=commandeMeteo1

#  CommandeI<EvenementI> commande = new MaillonDate(new MaillonHumidite(new MaillonPrecipitation(testCommande)));
bean.id.4=maillonPrecipitation
maillonPrecipitation.class=question3.MaillonPrecipitation
maillonPrecipitation.property.1=successeur
maillonPrecipitation.property.1.param.1=testCommande

bean.id.5=maillonHumidite
maillonHumidite.class=question3.MaillonHumidite
maillonHumidite.property.1=successeur
maillonHumidite.property.1.param.1=maillonPrecipitation

bean.id.6=chaineDesResponsables
```

```
chaineDesResponsables.class=question3.MaillonDate
chaineDesResponsables.property.1=successeur
chaineDesResponsables.property.1.param.1=maillonHumidite


bean.id.7=testCommande2
testCommande2.class=question4.Z_TestServiceLocator$TestCommande
testCommande2.property.1=nom
testCommande2.property.1.param.1=commandeMeteo2

#
bean.id.8=decorateurPrecipitation
decorateurPrecipitation.class=question2.DecorateurPrecipitation
decorateurPrecipitation.property.1=commande
decorateurPrecipitation.property.1.param.1=testCommande2

bean.id.9=decorateurHumidite
decorateurHumidite.class=question2.DecorateurHumidite
decorateurHumidite.property.1=commande
decorateurHumidite.property.1.param.1=decorateurPrecipitation

bean.id.10=decorateurs
decorateurs.class=question2.DecorateurDate
decorateurs.property.1=commande
decorateurs.property.1.param.1=decorateurHumidite


# A placer en dernier, contrainte due à l'initialisation des beans étant des variables d'instance
#                   d'autres beans, notamment les médiateur ci-dessous
bean.id.11=servicelocator
servicelocator.class=question4.ServiceLocator
servicelocator.property.1=mediateur
servicelocator.property.1.param.1=mediateur_1
servicelocator.property.2=mediateur
servicelocator.property.2.param.1=mediateur_2
```

- **La classe de tests unitaires fournie doit être lue attentivement et ne doit pas être modifiée et s'exécuter sans erreur.**

- **N'oubliez pas de soumettre cette question à l'outil JNEWS**

- N'oubliez pas de mettre votre nom au début de chaque classe que vous avez complétée
    - dans les commentaires au début de chaque classe /** @author MARTIN .... */

N'oubliez pas de déposer votre examen*

Avant 21h, via Bluej item submit, **attention à l'échéance de 21h** l'outil n'a pas de tolérance...
Enfin, **toujours avant 21h**, générez l'archive java de votre travail (sous Bluej menu Projet item Create Jar File/Exporter) puis

- déposez cette archive sur votre agenda, comme un tp ordinaire (sans rapport cette fois)
- **et avant 21h15** envoyez moi cette archive (ex: **MARTIN_ExamenFevrier2021.jar)** via WeTransfer (https://wetransfer.com/) à jean-michel.douin@lecnam.net, en indiquant vos nom/prénom/identifiant_Cnam, vous recevrez ainsi un accusé de la réception de votre travail

* **Une idée de la solution sera en ligne à cette URL : http://jfod.cnam.fr/NFP121/annales/2021_fevrier/**