Exercices pour révision pour l'examen intra
(Questions extraites d'examens intra de sessions passées)

Question 1

A) Que signifie la notion d'encapsulation? restreindre l'accès direct aux attributs d'une classe en les rendant privés et en fournissant des methodes publiques
B) Citez un patron qui renforce l'encapsulation et expliquez comment il la renforce. il fournit un iterateur qui cache la structure donnees

Question 2

Plusieurs patrons utilisent l'héritage de classe (ou l'implémentation d'interface) et le polymorphisme.

A) Utilisez un de ces patrons pour expliquer la notion de polymorphisme. médiateur
B) Expliquez quel impact cela a sur le couplage entre classes.

Question 3

Pour chacun des cas décrits ci-dessous :

- **Indiquez** le patron de conception qui devrait être appliqué.
- **Donnez** un avantage lié à l'utilisation de ce patron dans le contexte de cette application.

A) Différents courtiers utilisent une application qui analyse les côtes en bourse de diverses compagnies. Si une variation d'une côte survient, l'application doit permettre aux courtiers de déclencher différents processus selon la valeur de cette variation.
B) Dans une application de gestion des commandes, il y a différents types de commande (ex: commande régulière, commande expresse, commande spéciale, commande cadeau). Les différents types de commande doivent être traités en suivant un ensemble identique d'étapes. Cependant, certaines étapes du traitement peuvent différer selon le type de la commande. template

Question 4

La figure 1 présente une conception qui a été réalisée pour renseigner certains utilisateurs (une ferme et un aéroport) sur les conditions météo de la région.

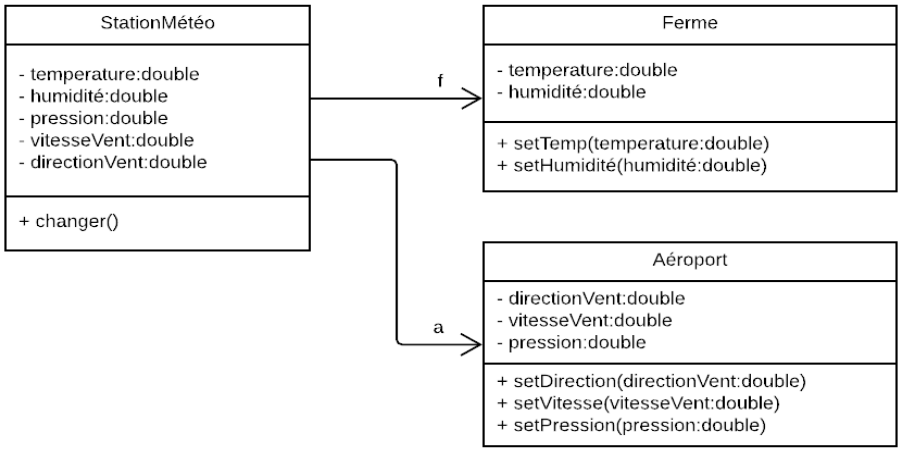


Figure 1

Comme on le voit, chaque utilisateur a des besoins d'informations différents. Par exemple, la ferme n'a besoin que de la température et de l'humidité.

1) La responsabilité de la méthode changer() de StationMétéo est de mettre à jour l'instance **f** de Ferme et l'instance **a** de Aéroport lorsqu'il y a un changement dans un des états de la station météo. En se basant sur le diagramme de classes de la figure 1, dessinez le diagramme de séquences qui représente le comportement de ce système lorsque la méthode changer() est appelée. Le diagramme de la figure 2 est une ébauche qui devrait servir de point de départ de votre diagramme de séquences.

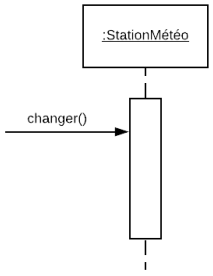


Figure 2

2) Une demande est faite pour modifier le système de la figure 1. En effet, un nouvel utilisateur, le club de voile, est aussi intéressé par certaines données de la station météo. Avant d'exécuter cette demande de modification, expliquez quels sont les problèmes que vous voyez avec la conception de la figure 1. Station connait tout le monde

3) On vous demande de revoir la conception présentée à la figure 1 afin de rendre la classe StationMétéo plus indépendante des classes qui utilisent ses informations. Vous pouvez faire des changements aux classes existantes. Indiquez le nom du parton de conception que vous

 Observateur

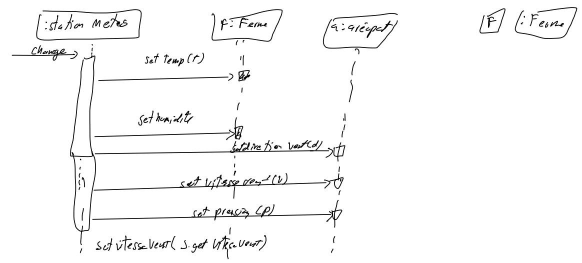
allez utiliser, et dessinez le diagramme de classes de votre solution en détaillant tous les éléments reliés à l'application du patron.

4) Supposons que votre nouvelle classe StationMétéo a une méthode setVitesseVent(vitesse :double) dans votre nouveau système, tel que vous l'avez conçu au point 3 de cette question. Faites le diagramme de séquences qui illustre le comportement du nouveau système quand la méthode setVitesseVent(vitesse :double) est appelée.

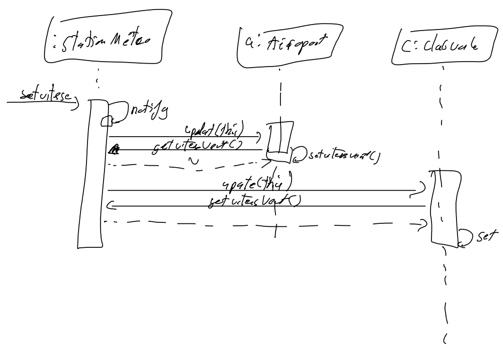
Question 5

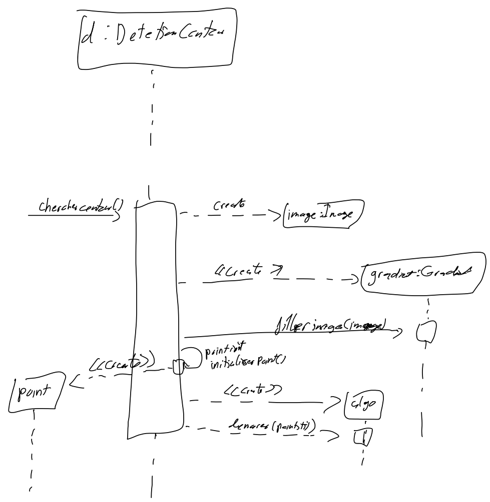
Considérons le code source suivant. Faite le diagramme de séquences correspondant à l'appel de la méthode `d1.chercherContour()`, où `d1` est une instance de la classe `DetectionContour`.

```
public class DetectionContour {

        public class Image {/*...*/}

        public class Point {/*...*/}

        public class Gradient {
          void filtrerImage(Image image) {/*...*/}
        }

        public class MarchingSquare  {
          void demarerRecherche(Point point) {/*...*/}
        }

        public Point initialiserPoint() {
                Point point = new Point();
                /*...*/
                return point;
        }

        public void chercherContour() {
                Image image = new Image();
                Gradient gradient = new Gradient();
                gradient.filtrerImage(image);
                Point pointInitial = initialiserPoint();
                MarchingSquare algo = new MarchingSquare();
                algo.demarerRecherche(pointInitial);
        }

        public static void main(String[] args){
                DetectionContour detecteur = new DetectionContour();
                detecteur.chercherContour();
        }
}
```


:station Météo
f: Ferme
a:aéroport
set temp (t)
set humidité
set pression (p)
set vitesseVent(s.get VitesseVent)


Sujet
-notify
Observer
<<int>>
Observateur
update()
station météo
Ferme
update()
temp = s.getTemp()
hum = s.getHumidité()
Vale


:station Météo
a: Aéroport
c: club voile
setvitesse
notify
update(this)
getvitesseVent()
update(this)
getvitesseVent()
set


d : DetectionCentre
checkcentre()
create
image:Image
<<create>>
point
<<create>>
algo

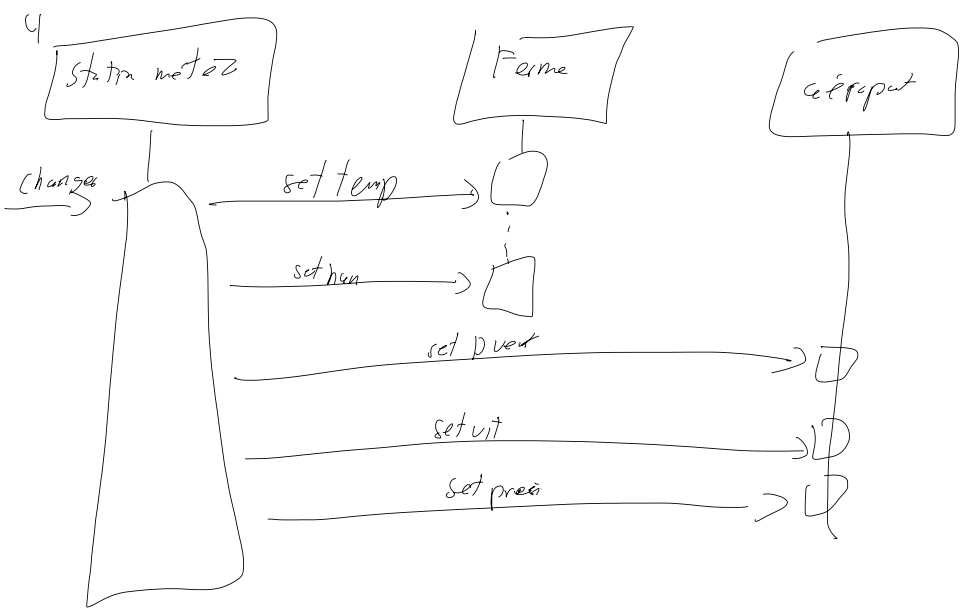
2

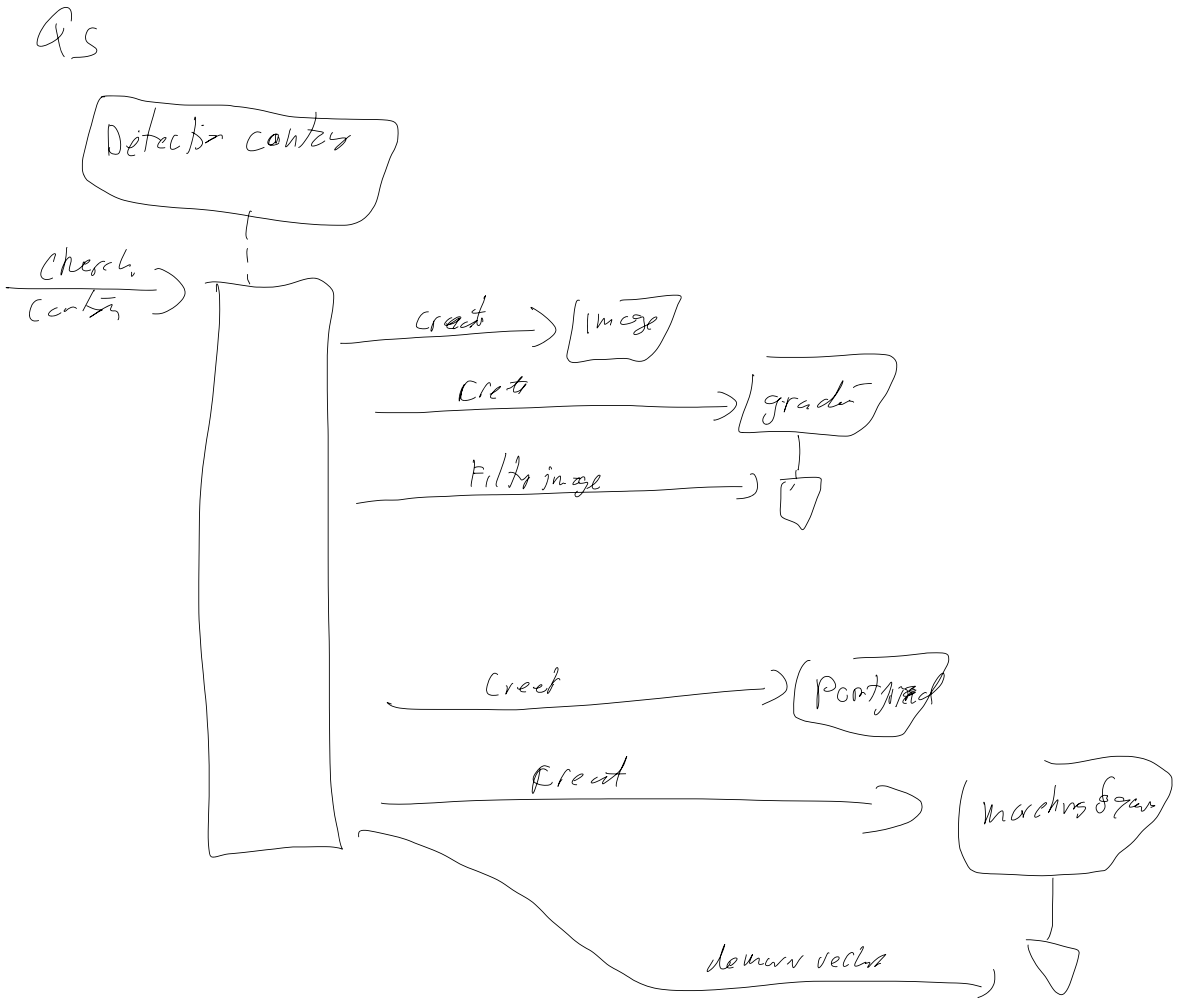
A) Le patron Médiateur et le polymorphisme

Le patron Médiateur utilise le polymorphisme en définissant une interface commune pour centraliser les communications entre objets, permettant ainsi à différents composants de collaborer sans dépendre directement les uns des autres.

B) Impact sur le couplage

Il réduit le couplage entre classes en supprimant les dépendances directes entre objets, rendant le système plus modulaire et plus facile à modifier ou étendre.




Qs
Detection contour
Check contour
Create
Image
Create
gradient
Filter image
Create
Portrait
Create
Machine learning
Return vector