TP N°4

Classes Abstraites – Interfaces

Licence : INFO-AP | G-INFO

Université Moulay Ismaïl – Faculté des Sciences Meknès

Pr. MAROUANE NAZIH & Pr. ISMAILI ALAOUI ELMEHDI

Énoncé du TP : Gestion d'une flotte de véhicules intelligents

Ce TP vous plonge dans la modélisation d'une flotte de véhicules modernes : thermiques, électriques et autonomes.

Vous appliquerez les concepts essentiels de la POO (héritage, abstraction, interfaces, polymorphisme) pour construire une architecture claire et évolutive.

L'objectif est de comprendre comment organiser un système capable d'intégrer plusieurs types de comportements tout en restant cohérent.

Travail demandé

1. Classe abstraite `Vehicule`

Créer une classe abstraite `Vehicule` avec les attributs suivants :

— `immatriculation : String`

— `marque : String`

Travail à réaliser :

— Constructeur d'initialisation,

— Méthode concrète `afficherInfos()` affichant les deux attributs,

— Méthode abstraite `double calculerAutonomie()`.

2. Interface `Connectable`

Creer une interface `Connectable` contenant deux methodes (contrats obligatoires). Toute classe qui implemente cette interface devra obligatoirement redefinir ces methodes et afficher les phrases suivantes pour simuler leur fonctionnement.

— `void connecterServeur();`
Affiche : `"Connexion au serveur central..."`

— `void envoyerDiagnostic();`
Affiche : `"Diagnostic envoye au serveur."`

3. Interface `Autonome`

Creer une interface `Autonome` contenant deux methodes (contrats obligatoires). Toute classe qui implemente cette interface devra redefinir ces methodes et produire les messages suivants.

— `void activerPilotageAuto();`
Affiche : `"Pilotage automatique active."`

— `int niveauAutonomie();`
Retourne un entier (1 a 5) et affiche par exemple :
`"Niveau d'autonomie actuel : X"`

4. Classe `VoitureThermique`

Créer une classe `VoitureThermique` héritant de `Vehicule`.
Attributs :

- `capaciteReservoir : double`
- `consommationMoyenne : double`

Travail demandé :

- Constructeur complet,
- Redéfinition de `afficherInfos()` pour ajouter les détails du véhicule thermique.
- Redéfinition de `calculerAutonomie()` :

$$autonomie = \frac{capaciteReservoir}{consommationMoyenne} \times 100$$

5. Classe `VoitureElectrique`

Créer une classe `VoitureElectrique` héritant de `Vehicule` et implémentant `Connectable`.
Attributs :

- `capaciteBatterie : double`
- `consoKWhPar100Km : double`

Travail demandé :

- Constructeur complet,
- Redéfinition de `afficherInfos()` pour ajouter les détails du véhicule Electrique.
- Redéfinition de `calculerAutonomie()` :

$$autonomie = \frac{capaciteBatterie}{consoKWhPar100Km} \times 100$$

- Implémentation de `connecterServeur()` et `envoyerDiagnostic()`.

6. Classe `CamionAutonomeConnecte`

Créer une classe `CamionAutonomeConnecte` héritant de `Vehicule` et implémentant `Connectable` et `Autonome`.
Attributs :

- `capaciteReservoir : double`
- `niveauIA : int`

Travail demandé :

- Constructeur complet,
- Redéfinition de `afficherInfos()` pour ajouter les détails du Camion Autonome Connecté.
- Redéfinition de `calculerAutonomie()` (modèle simple) :

$$autonomie = capaciteReservoir \times 3 \times (1 + 0.02 \times niveauIA)$$

- Implémentation des méthodes de `Connectable`,
- Implémentation des méthodes de `Autonome`.

Étude de cas : Classe de test `TestFlotte`

Creer une classe `TestFlotte` contenant une methode `main()` qui execute le scenario suivant :

```
public class TestFlotte {
    public static void main(String[] args) {

        // 1. Creation d'un tableau de Vehicule
        Vehicule[] flotte = new Vehicule[] {
            new VoitureThermique("1234-AB-01", "Peugeot", 50.0, 6.5),
            new VoitureElectrique("5678-CD-02", "Renault", 40.0, 15.0),
            new CamionAutonomeConnecte("9012-EF-03", "Volvo", 300.0, 4)
        };

        // 2. Parcours du tableau
        for (Vehicule v : flotte) {
            System.out.println("----------------------------------");

            // a) Afficher les informations du vehicule
            v.afficherInfos();

            // b) Afficher l'autonomie calculee
            System.out.println("Autonomie estimee : "
                               + v.calculerAutonomie() + " km");

            // c) Verifier si le vehicule est Connectable
            if (v instanceof Connectable) {
                Connectable c = (Connectable) v;
                c.connecterServeur();
                c.envoyerDiagnostic();
            }

            // d) Verifier si le vehicule est Autonome
            if (v instanceof Autonome) {
                Autonome a = (Autonome) v;
                a.activerPilotageAuto();
                System.out.println("Niveau autonomie : "
                                   + a.niveauAutonomie());
            }
        }
    }
}
```