Travaux pratiques N°3

Chapitres : Héritage, Polymorphisme, classes Abstraites, Interfaces, et gestion des exceptions

Exercice 1 : Robot

On considère la hiérarchie de classes **Robot**, **RobotExtra**, **RobotBissectrice** et **RobotExtraNG**. Ces classes modélisent l'état et le comportement de robots virtuels. La classe abstraite **Robot** est un modèle qui définit les robots comme suit :

Chaque robot :

- A un nom (l'attribut protégé **nom** : chaîne de caractères).
- A une position (x,y) : donnée par les deux attributs protégés et entiers **x** et **y**.
- A une direction : donnée par l'attribut protégé **direction** qui prend une des valeurs **"Nord"**, **"Est"**, **"Sud"**, **"Ouest"**, **"NO"** (le nord-ouest), **"NE"** (le nord-est), **"SO"** (le sud-ouest), et **"SE"** (le sud-est).
- Peut avancer en avant avec la méthode abstraite : **void avance( )**.
- Peut tourner à droite avec la méthode abstraite : **void rotation( )**. Les robots ne peuvent pas tourner à gauche.
- Peut afficher son état avec la méthode sans paramètre : **void afficher( )**.

Le nom, la position et la direction d'un robot lui sont donnés au moment de sa création. Le nom est obligatoire mais on peut ne pas spécifier la position et la direction, qui doivent être définis par défaut à **(0,0)** et **"Est"** (On utilise deux constructeurs pour la classe **Robot**).

1) Ecrire les instructions Java qui permettent de définir la classe **Robot**.

Les **RobotExtra** se comportent selon le modèle **Robot**, et chaque robot de cette classe :

- Peut avancer suivant sa **direction d'un pas (u ou v)** en avant (Dans l'exemple du schéma : le robot R dont la direction actuelle est **"Est"** avance d'un pas **v** de la position $R^{(1)}$ à la position $R^{(2)}$. Le robot S dont la direction actuelle est **"SE"** avance d'un pas **u** de la position $S^{(1)}$ (4,-3) à la position $S^{(2)}$(5,-4)…) : les robots peuvent avancer avec la méthode à implémenter **void avance( )** (On utilise la structure ***switch(…){…}***).
- Peut tourner à droite de 45° pour changer de direction avec la méthode à implémenter **void rotation( )** (On utilise la structure ***switch(…){…}***).

2) Ecrire les instructions Java qui permettent de définir la classe **RobotExtra**.

Les **RobotBissectrice** sont des **Robot**s particuliers dans la mesure où ils se déplacent uniquement sur la droite ***(D): y = x*** et chaque robot de cette classe :

- Doit avancer si sa **direction** le permet vers la position symétrique par rapport à l'origine (0,0). Si sa direction ne permet pas ce type de déplacement, il se contente de se translater suivant l'un des vecteurs **u(1,1)** ou **u(-1,-1)** (Dans l'exemple du schéma ci-dessous : le robot P dont la direction actuelle est **"SO"** avance de la position $P^{(1)}$(2,2) à la position symétrique $P^{(2)}$ (-2,-2). Le robot Q dont la direction actuelle est **"NE"** avance de la position $Q^{(1)}$(4,4) à la position $Q^{(2)}$ (5,5)…) : les robots peuvent avancer avec la méthode à implémenter **void avance( )**.
- Peut tourner à droite de 180° pour changer de direction avec la méthode à implémenter **void rotation( )**.
- Peut lever une exception de type **DirectionException** si l'utilisateur saisisse une direction différente des 2 directions admissibles.

3) Ecrire les instructions Java qui permettent de définir la classe **RobotBissectrice**.

On veut améliorer les robots de type **RobotExtra** en créant une Nouvelle Génération, les **RobotExtraNG** qui ne remplacent pas les anciens robots mais peuvent cohabiter avec eux.

Les **RobotExtraNG** savent faire la même chose mais aussi activer un mode « *turbo* » et le désactiver. Dans ce mode, les robots avancent selon le parcours à 3 étapes suivant : 1. Avancer comme un **RobotExtra** puis 2. Faire une rotation à droite et finalement 3. Avancer une autre fois comme un **RobotExtra**.

Ces robots peuvent aussi :

- Répéter le parcours **n** fois grâce à une méthode **avance(int n)** qui prend en paramètre le nombre de fois de parcours.
- Tourner à gauche de **45°** grâce à la méthode **rotationGauche( )**.
- Faire demi-tour grâce à la méthode **demiTour( )**.

Les nouvelles méthodes doivent appeler les anciennes méthodes pour implémenter le nouveau comportement.

Dans la classe **RobotExtraNG**, il faut :

- Définir un attribut **turbo** pour stocker l'état du mode *turbo* du robot (le mode *turbo* doit être désactivé par défaut).
- Redéfinir la méthode sans paramètres **avance( )** de la classe **RobotExtra** de sorte que le mode *turbo* soit pris en compte.
- Ecrire la méthode **avance (int n)** en prenant en compte le mode *turbo*(activé/désactivé).
- Ecrire une méthode **setTurbo(…)** qui permet d'activer/désactiver le mode *turbo*.
- Ecrire une méthode **isTurbo( )** qui permet d'indiquer si le mode *turbo* est actif.
- Redéfinir la méthode **afficher( )** pour ajouter l'état actuel du mode *turbo*.

4) Ecrire cette nouvelle classe en **spécialisant** la classe **RobotExtra**, sans modifier celle-ci.

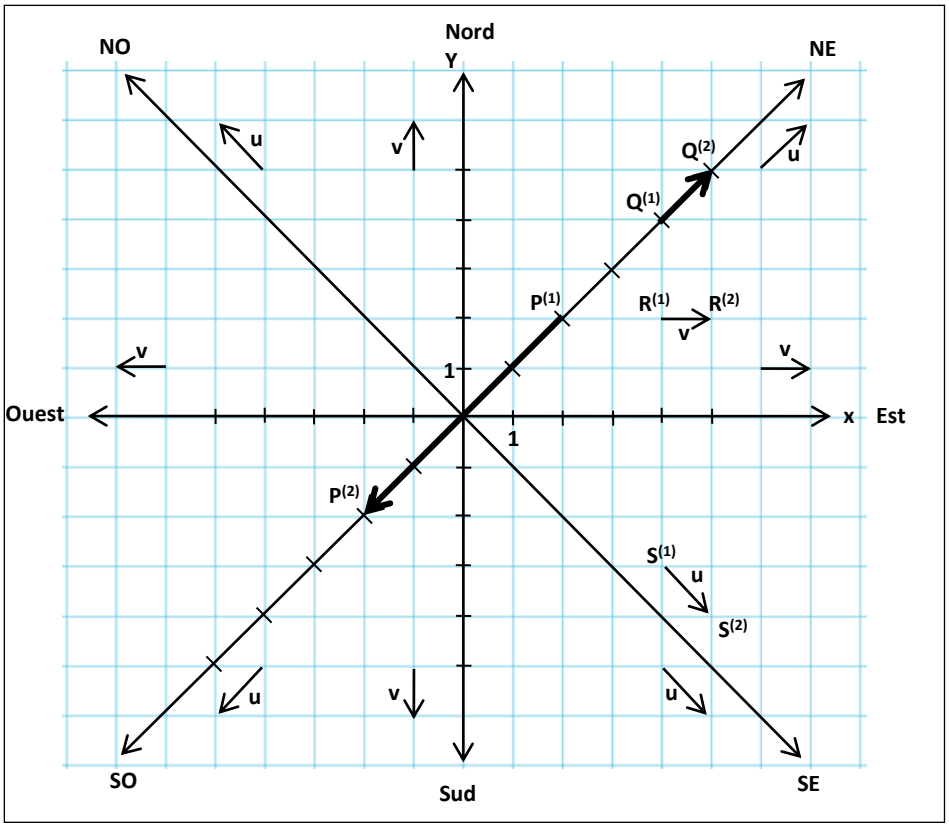
5) Dans une classe **RobotTest** contenant la méthode **main(…)** créer un tableau de 3 objets de type **Robot** qui doit contenir
un robot *[Danel ;(7,7) ;NE]* de type **RobotBissectrice**, 2 robots *[Bonbon ;(6,-10) ;Nord]* et *[Sinsodine ;(15,-15) ;SO]* de type **RobotExtraNG**.

5.1. Appliquer au robot *Danel* les méthodes *rotation(), avance()* puis *avance()* et *afficher()* (Gérer l'exception **DirectionException**).

5.2. Appliquer au robot *Bonbon* les méthodes *rotationGauche(), demiTour(), avance(7)* et *afficher()*.

5.3. Après avoir activé le mode *turbo*, appliquer au robot *Sinsodine* les méthodes *rotationGauche(), avance(3)* et *afficher()* (Une conversion explicite de type est nécessaire pour les robots **RobotExtraNG**).

5.4. Donner les positions et les directions des 3 robots après l'exécution.



Exercice 2 : Bibliothèque

L'objectif est d'écrire une application traitant des documents de nature diverse : des livres, qui peuvent être des romans ou des manuels, des revues, des dictionnaires, etc…

- Tous les documents ont un numéro d'enregistrement (un entier) et un titre (une chaîne de caractères). Les livres ont, en plus, un auteur (une chaîne) et un nombre de pages (un entier).
- Les romans ont éventuellement un prix littéraire (un entier conventionnel, parmi : GONCOURT, MEDICIS, INTERALLIE, etc.), tandis que les manuels ont un niveau scolaire (une chaîne).
- Les revues ont un mois et une année (des entiers).
- Les dictionnaires ont une langue (une chaîne de caractères convenue, comme "anglais", "allemand", "espagnol", etc.).
- Tous les divers objets en question ici (livres, revues, dictionnaires, romans, etc.) doivent pouvoir être manipulées en tant que documents.

1- Définir les classes **Document**, **Livre**, **Roman**, **Manuel**, **Revue** et **Dictionnaire**, entre lesquelles existeront des liens d'héritage. Dans chacune de ces classes définir.
   - le constructeur qui prend autant arguments qu'il y a de variables d'instance et qui se limite à initialiser ces dernières avec les valeurs des arguments, une méthode public String **toString()** produisant une description sous forme de chaîne de caractères des objets,
   - Définir également des « accesseurs » publics get... permettant de consulter les valeurs de ces variables.
   - Écrire une classe exécutable **TestDocuments** qui crée et affiche plusieurs documents de types différents.

2. Une bibliothèque sera représentée par un tableau de Documents. Définissez une classe **Bibliotheque**, avec un tel tableau pour variable d'instance et les méthodes :
   - Bibliotheque(int capacité) : constructeur qui crée une bibliothèque ayant la capacité (nombre maximum de documents) indiquée,
   - void afficherDocuments() : affiche tous les ouvrages de la bibliothèque,
   - Document document(int i) : renvoie le ième document,
   - boolean ajouter(Document doc) : ajoute le document indiqué et renvoie true (false en cas d'échec),
   - boolean supprimer(Document doc) : supprime le document indiqué et renvoie true (false en cas d'échec)
   - void afficherAuteurs() : affiche la liste des auteurs de tous les ouvrages qui ont un auteur (au besoin, utilisez l'opérateur instanceof).

3. Définir, une classe **Livrotheque** dont les instances ont les mêmes fonctionnalités que les Bibliotheques mais sont entièrement constituées de livres.

Cette classe hérite des attributs et des méthodes de la classe Bibliotheque. Elle possède un constructeur qui prend comme argument la capacite et les méthodes suivantes :

- public boolean ajouter(Document doc)
- public Livre livre(int i)
- public void afficherAuteurs

Comment optimiser dans la classe Livrotheque la méthode afficherAuteurs ?

Exercice 3 : Centre de soutien scolaire

Un centre de soutien scolaire propose des heures supplémentaires aux élèves du lycée afin d'améliorer leur niveau scolaire en trois matières scientifiques selon trois niveaux. Les prix (en Dh) sont affichés dans la table suivante :

Matière	**Niveau 1** **Tronc Commun**	**Niveau 2** **1 Année Bac**	**Niveau 3** **2 Année Bac**
Sciences Mathématiques	1500	2500	3500
Sciences Physiques	1300	1500	3000
Sciences de la vie et de la terre	1300	1500	3000

Les différents attributs des classes sont privés.

1. Ecrire une classe nommée **Matiere** :
   - qui contient une description concernant les différentes matières enseignées et leurs frais d'enseignement. Inclure un constructeur pour l'initialisation des différents attributs.
   - Redéfinir la méthode **toString()** pour afficher les informations sur une matière comme suit :
     Mathématiques -> (Niveau 1 : 1500 Dh), (Niveau 2 : 2500 Dh), (Niveau 3 : 3500 Dh)
2. Ecrire la classe abstraite **Personne** qui contient :
   - les attributs ***nom*** et ***prenom***. Inclure un constructeur pour l'initialisation des différents attributs.
   - la méthode abstraite ***statut( )*** prévue pour afficher le nom suivi du statut de chaque personne.
3. Ecrire la classe **Eleve** qui hérite de **Personne**. Un élève est inscrit dans un niveau d'une seule matière. Gérez l'exception si l'utilisateur saisisse un niveau différent de 1, 2 ou 3.
4. Ecrire la classe **Enseignant** qui hérite de **Personne**. Un enseignant enseigne une seule matière.
5. Ecrire la classe **Directeur** qui hérite de **Personne**. Un directeur dispose d'un nom de connexion et d'un mot de passe.
6. Ecrire une classe **CentreDeSoutien** contenant la méthode **main()** qui doit respecter l'exemple d'exécution ci-dessous. Dans cette méthode il faut :
   - déclarer un tableau qui contient 3 objets de type Matière et remplir le tableau par les données de la table.
   - déclarer un tableau de trois enseignants (un enseignant par matière).
   - déclarer un tableau de trois élèves (un élève par matière et par niveau).
   - afficher le statut des différentes personnes.
   - afficher les différentes matières ainsi que leurs informations.
   - demander à l'utilisateur de faire un choix (Mathématiques : 0, Physique : 1 et Science de la vie : 2) et afficher les informations concernant la matière choisie.
   - Gérer les différentes exceptions relatives à cette opération de saisie. Utiliser InputMismatchException (si ce n'est pas un entier) et **ArrayIndexOutOfBoundsException** (si c'est en dehors du tableau).

Exemple d'exécution :

```
---------------------------------------- Liste des Personnes ----------------------------------------
Nom :Alaoui Mohammed, Statut : Directeur
Nom :Merbah Imad, Statut : Enseignant
Nom :Zinan Sara, Statut : Enseignant
Nom :Mbarek Anas, Statut : Enseignant
Nom :Maknasi Aziza, Statut : Elève
Nom :Chiban Aziz, Statut : Elève
Nom :Rmili Redouane, Statut : Elève
---------------------------------------- Liste des Matières ----------------------------------------
Sciences Mathématiques --> (Niveau 1 : 1500Dh), (Niveau 2 : 2500Dh), (Niveau 3 : 3500Dh)
Sciences Physiques --> (Niveau 1 : 1300Dh), (Niveau 2 : 1500Dh), (Niveau 3 : 3000Dh)
Sciences de la vie et de la terre --> (Niveau 1 : 1300Dh), (Niveau 2 : 1500Dh), (Niveau 3 : 3000Dh)
------------------------------------------------ Choix ------------------------------------------------
Mathèmatiques : 0
Physique : 1
Science de la vie : 2
Veuillez saisir votre choix : 2
Sciences de la vie et de la terre --> (Niveau 1 : 1300Dh), (Niveau 2 : 1500Dh), (Niveau 3 : 3000Dh)
```

Exercice 4 : Entreprise

Le directeur d'une entreprise de produits chimiques souhaite gérer les salaires et primes de ses employés au moyen d'un programme Java.

1- Coder une classe abstraite **Employe** dotée des attributs nécessaires (Un employé est caractérisé par son nom, son prénom, son âge et sa date d'entrée en service dans l'entreprise), d'une méthode abstraite **calculerSalaire** (ce calcul dépendra en effet du type de l'employé) et d'une méthode **getNom** retournant une chaine de caractère obtenue en concaténant la chaine de caractères "L'employé " avec le prénom et le nom.
Ajouter également à la classe un constructeur prenant en paramètre l'ensemble des attributs nécessaires.

Le calcul du salaire mensuel dépend du type de l'employé. On distingue les types d'employés suivants :

- Ceux affectés à la ***Vente***. Leur salaire mensuel est le 20 % du *chiffre d'affaire* qu'ils réalisent mensuellement, plus 4000 dhs.
- Ceux affectés à la ***Représentation***. Leur salaire mensuel est également le 20 % du *chiffre d'affaire* qu'ils réalisent mensuellement, plus 8000 dhs.
- Ceux affectés à la ***Production***. Leur salaire vaut le *nombre d'unités* produites mensuellement multipliées par 5.
- Ceux affectés à la ***Manutention***. Leur salaire vaut leur *nombre d'heures* de travail mensuel multipliées par 650 dhs.

2- Coder une hiérarchie de classes pour les employés en respectant les conditions suivantes :

- La super-classe de la hiérarchie doit être la classe **Employe**.
- Les nouvelles classes doivent contenir les attributs qui leur sont spécifiques ainsi que le codage approprié des méthodes **calculerSalaire** et **getNom**, en changeant le mot "employé" par la catégorie correspondante.
- Chaque sous classe est dotée de constructeur prenant en argument l'ensemble des attributs nécessaires.
- N'hésitez pas à introduire des classes intermédiaires pour éviter au maximum les redondances d'attributs et de méthodes dans les sous-classes.

Certains employés des secteurs *production* et *manutention* sont appelés à fabriquer et manipuler des produits dangereux.
Après plusieurs négociations syndicales, ces derniers parviennent à obtenir une prime de risque mensuelle.

3- Compléter votre programme introduisant deux nouvelles sous-classes d'employés. Ces sous-classes désigneront les employés des secteurs *production* et *manutention* travaillant avec des produits dangereux.
Ajouter également à votre programme une interface pour les *employés à risque* permettant de leur associer une *prime mensuelle* fixe de 2000.

Satisfait de la hiérarchie proposée, le directeur souhaite maintenant l'exploiter pour afficher le salaire de tous ses employés ainsi que le salaire moyen.

4- Ajouter une classe Personnel contenant une "collection" d'employés. Il s'agira d'une collection polymorphique d'Employe (Tabelau ou liste). Définissez ensuite les méthodes suivantes à la classe Personnel :

- voidajouterEmploye(Employe) : qui ajoute un employé à la collection.
- voidcalculerSalaires() : qui affiche le salaire de chacun des employés de la collection.
- double salaireMoyen() : qui affiche le salaire moyen des employés de la collection.

5- Tester votre programme avec le main suivant :

```
public class Salaires {
	public static void main(String[] args){
	Personnel p =new Personnel();
	p.ajouterEmploye(new Vendeur("Mohammed", "Kamil", 45, "1995", 300000));
	p.ajouterEmploye(new Representant("Tayeb", "Amrani", 25, "2001", 200000));
	p.ajouterEmploye(new Technicien("Sara", "Kamili", 28, "1998", 1000));
	p.ajouterEmploye(new Manutentionnaire("Amine", "Moukhtari", 32, "1998", 45));
	p.ajouterEmploye(new TechnARisque("Anas", "Alaoui", 28, "2000", 1000));
	p.ajouterEmploye(new ManutARisque("Kamal", "Sabir", 30, "2001", 45));

	p.afficherSalaires();
System.out.println("Le salaire moyen dans l'entreprise est de " +p.salaireMoyen()+" Dhs.");
	}
}
```

Exemple d'exécution :

```
Le vendeur Mohammed Kamil gagne 64000.0 dhs.
Le représentant Tayeb Amrani gagne 48000.0 dhs.
Le technicien Sara Kamili gagne 5000.0 dhs.
Le manutentionnaire Amine Moukhtari gagne 2925.0 dhs.
Le technicien Anas Alaoui gagne 7000.0 dhs.
Le manutentionnaire Kamal Sabir gagne 4925.0 dhs.
Le salaire moyen dans l'entreprise est de 21975.0 Dhs.
```