**Université de Carthage**

**Institut National des Sciences Appliquées et de Technologie**



**Rapport de projet Java :**

**Application de Gestion De Robot**

**Elaboré par :**

**Aziza Garbâa**

**Annee universitaire :**

**2024/2025**

1. **Introduction**

Ce projet consiste à développer une application de gestion de robots capables d'effectuer diverses tâches (livraison, connexion réseau, etc.) tout en préservant l’environnement et en réduisant leur impact écologique.

La réalisation s’appuie sur les principes de la programmation orientée objet tels que l’héritage, le polymorphisme et la redéfinition de méthodes, afin d’assurer une architecture claire, modulable et évolutive.

1. **Diagramme de classe**

Une image contenant texte, diagramme, Plan, Dessin technique

Le contenu généré par l’IA peut être incorrect.

1. **Application des concepts de la POO**
2. **Héritage :**

Dans ce projet, j’ai appliqué le principe d’héritage pour définir des comportements communs à plusieurs types de robots, ce qui permet de réduire la duplication de code et de favoriser la réutilisabilité. La classe Robot, qui est une classe abstraite, contient les méthodes de base telles que demarrer(), arreter(), consommerEnergie(), ainsi que plusieurs méthodes abstraites qui doivent être redéfinies par les classes filles. La classe RobotConnecté hérite de Robot et ajoute des fonctionnalités spécifiques, comme la connexion à un réseau. Elle ne les redéfinit pas les méthodes abstraite de la classe Robot. Par conséquent, elle reste également abstraite. Enfin, la classe RobotLivraison hérite de RobotConnecté (et donc aussi de Robot) et redéfinit les méthodes abstraites . Elle devient donc concrète, et peut être instanciée pour créer un robot de livraison doté de toutes les fonctionnalités précédentes, ainsi que de comportements spécifiques liés à la gestion des colis et à la livraison.

1. **Encapsulation :**

Le principe de l’encapsulation est appliqué à travers la gestion de la visibilité des attributs. Dans toutes les classes du projet, les attributs sont déclarés comme privés (private) afin d'en protéger l’accès direct depuis l’extérieur. Pour y accéder ou les modifier, des méthodes d’accès (getters) et de modification (setters) sont mises en place. Cela garantit un meilleur contrôle sur les données internes des objets, en respectant les règles de sécurité et de cohérence de l’état des objets.

1. **Polymorphisme :**

Le polymorphisme par surcharge (ou surcharge de méthode) consiste à définir plusieurs méthodes avec le même nom dans une même classe, mais avec des signatures différentes (c’est-à-dire un nombre ou des types de paramètres différents).  
Cela permet d’adapter le comportement d'une méthode en fonction des données disponibles tout en gardant une cohérence dans la dénomination des actions réalisées. Dans ce projet, ce principe est appliqué à la méthode faireLivraison(). Elle est définie de deux manières : - Une version avec paramètres, utilisée pour effectuer une livraison simple vers une destination donnée (avec les coordonnées). - Une version sans paramètres, utilisée pour effectuer une livraison groupée de plusieurs colis. Ainsi, le même nom de méthode est conservé pour deux scénarios proches, ce qui rend le code plus lisible et plus logique.

1. **Gestion des exception :**

Dans ce projet, la gestion des exceptions a été utilisée pour mieux contrôler les erreurs possibles pendant l’exécution du programme. Une classe RobotException a été créée, héritant de la classe Exception, ce qui en fait une exception vérifiée. Ensuite, deux autres classes, EnergieInsuffisanteException et MaintenanceRequiseException, héritent de RobotException pour gérer des cas spécifiques.

La méthode deplacer() fait partie du mécanisme de gestion des exceptions : elle peut effectivement lancer une exception, d'où l'utilisation du mot-clé throws. Cette exception est ensuite propagée à travers les méthodes faireLivraison() et effectuerTache(), jusqu’à atteindre la méthode main dans la classe Test, où elle est capturée grâce à un bloc try-catch. Cela permet de gérer proprement les erreurs et d’éviter que le programme ne s’arrête brutalement..

1. **Interface :**

Dans cette application, deux interfaces principales sont utilisées pour définir des comportements clés des robots :

* Connectable : Cette interface contient les méthodes abstraites connecter(), deconnecter() et envoyerDonnees(). Ces méthodes sont implémentées dans la classe RobotConnecté, ce qui permet de gérer la connexion et l'envoi de données pour les robots connectés à un réseau.
* Runnable : Cette interface contient une seule méthode, run(), qui est cruciale pour le multithreading. Dans la classe RobotConnecté, la méthode run() est redéfinie pour permettre une gestion dynamique de la connexion. Elle permet d’interrompre temporairement la connexion et de rendre le port de connexion "down" lorsque le robot n’envoie pas de données. Un thread est créé pour chaque robot, et ce thread vérifie en permanence l'état du port de connexion. Si le port est "up" mais qu'aucune donnée n'est envoyée, il le met en état "down".

Ce mécanisme est particulièrement important dans un contexte écologique. En effet, lorsque la machine est connectée, elle émet des ondes nocives ayant des effets négatifs sur l’environnement. Grâce à cette gestion de la connexion, le temps d'émission des ondes est réduit au minimum, contribuant ainsi à une réduction de l'impact environnemental.

1. **Interface Graphique :**

L'interface graphique de l'application permet de gérer plusieurs robots simultanément grâce à l'utilisation d'un JTabbedPane, qui fournit plusieurs onglets. Chaque onglet représente un robot spécifique, ce qui permet de gérer différents robots en parallèle.

* Premier onglet : "Home"  
  Le premier onglet, intitulé "Home", contient un bouton de création d'un nouveau robot. Lorsqu'un utilisateur clique sur ce bouton, un ActionListener est activé, instanciant un objet de type RobotLivraison. Un thread est alors lancé pour ce robot, et un nouvel onglet spécifique à ce robot est ouvert.
* Structure de chaque onglet de robot:  
  Chaque onglet est divisé en deux parties grâce à un JSplitPane :
  1. Partie gauche : Cette section contient une grille où le robot se déplace. Un objet de type GridPanel est instancié ici. Cette classe, qui est une sous-classe personnalisée de JPanel, est responsable de l'affichage de la grille et du déplacement du robot.
     + Dans son constructeur, l'image du robot est chargée sous forme d'ImageIcon.
     + La méthode setRobotPosition() est définie pour déplacer le robot dans la grille à chaque livraison, et elle fait appel à la méthode repaint() pour redessiner la grille avec la nouvelle position du robot.
     + La méthode paintComponent() est redéfinie pour dessiner la grille et afficher le robot à sa position actuelle. Elle détermine la taille des cellules de la grille et dessine le robot à l'emplacement approprié.
     + La méthode getPreferredSize() est également redéfinie pour spécifier la taille préférée du panneau de la grille.
  2. Partie droite : Cette section est également divisée en deux parties :
     + Partie haute : Cette zone contient des boutons qui effectuent différentes actions, telles que "Livrer". Lorsqu'un bouton est cliqué, il appelle la méthode FaireLivraison(), initiant l'action de livraison pour le robot.
     + Partie basse : Il s'agit d'une zone d'affichage (un JTextArea) qui affiche l'historique des actions effectuées par le robot, telles que les livraisons réalisées, les erreurs, etc.

Ainsi, l'interface graphique permet non seulement de créer et de gérer plusieurs robots, mais aussi de visualiser en temps réel leurs actions et leur position dans la grille. Chaque robot a son propre onglet, et les actions spécifiques à chaque robot sont affichées de manière claire et organisée.

1. **Stratégies écologiques dans le fonctionnement des robots**

Pour des raisons écologiques, un mode éco a été défini dans la classe Robot. Ce mode modifie le comportement du robot en matière de chargement et de livraison afin de réduire la consommation d’énergie. Concrètement, lorsqu’il est activé, les colis dont les destinations sont proches sont regroupés, permettant ainsi de les livrer en une seule tournée et d’optimiser les déplacements.

Par ailleurs, afin de réduire l’émission d’ondes électromagnétiques nocives lors de la connexion réseau, le port de communication est temporairement désactivé lorsqu’aucune donnée n’est transmise.

Enfin, la classe Robot intègre un mécanisme de calcul de la quantité de CO₂ émise en fonction de l’énergie électrique consommée, permettant ainsi de suivre et maîtriser l’impact environnemental du fonctionnement des robots.

1. **Conclusion**

Ce projet de gestion de robots met en œuvre les principes fondamentaux de la programmation orientée objet, tout en intégrant une dimension écologique. Grâce à l’héritage, au polymorphisme et à la gestion des exceptions, l’application est à la fois flexible et robuste. L’ajout du mode éco et des mécanismes de réduction d’impact environnemental montre l’importance d’une conception responsable dans le développement logiciel moderne.