Programmation orientée objet en Java

Projet

Raúl MAZO PENA

Année scolaire 2025-2026

Indications pour bien réussir votre projet

Recommandations et consignes pour la formation des groupes et l'organisation du travail

1. Veuillez faire des **groupes de 4 étudiants**.
2. Le **mode Agile s'invite ici** pour éviter l'effet tunnel et pour vous aider à livrer des versions exécutables dès le début !
3. Donnez-vous les moyens de travailler correctement. Prenez le temps de vous organiser et de travailler en utilisant les outils qu'il vous faut : un bon IDE (e.g., IntelliJ Ultimate), un outil de gestion de versions et de travail collaboratif (Git et GitHub) …
4. Les **difficultés avec votre ordinateur**, **vos outils** (performances, configuration, bugs, licences, ...) **et votre équipe de travail** (horaires de travail, répartition des responsabilités, manque de confiance, manque d'engagement et de communication ...) **sont de la responsabilité de l'étudiant ou de l'équipe et NON des enseignants**.

Recommandations et consignes pour la réalisation du projet

1. **Réfléchissez bien avant de coder** : attention à la modélisation, au découpage, à la « duplication » de code, … Ceci est un cours de PROGRAMMATION ORIENTEE OBJET et non de programmation impérative.
2. **Testez**, testez et... testez encore (mieux : écrivez des tests !).
3. Pensez à **rédiger une javadoc** parfaite (la compiler et la relire). La javadoc ne fait que "formater" la documentation que vous avez renseignée dans le code source. Cela ne sera pas magique si vous ne documentez pas bien les classes, méthodes, paramètres, les exceptions propagées, … de votre application. Voici un tutoriel présentant l'outil Javadoc de Sun, qui permet de générer les documentations d'un code Java : https://simonandre.developpez.com/tutoriels/java/presentation-javadoc/
4. Lorsque vous êtes satisfaits de votre application, prenez le temps, à tête reposée, de **relire tout le code**. Interrogez-vous, étonnez-vous, vérifiez la cohérence entre les classes, les éventuelles redondances de code, et le besoin d'abstraire ou de rendre générique, réfléchissez aux éventuelles modifications futures et à leurs répercussions sur l'évolution du code, ...
5. Je vous conseille ensuite de **soumettre votre code à Sonarcube (et/ou Codacy)** ; ça devrait vous permettre d'encore améliorer votre code. Ces outils vous permettront de faire une analyse statique qui a pour but de mesurer la qualité du code de vos applications et de vous fournir des métriques portant sur la qualité du code et permettant d'identifier précisément les points à corriger (code mort, bugs potentiels, non-respect des standards, manque ou excès de commentaires...). L'utilisation de ce type d'outils ne remplace en aucun cas les tests unitaires, mais permet d'identifier rapidement certains défauts du code.

Recommandations pour la partie rapport et rendu du travail

Voici quelques recommandations pour préparer et rendre vos projets :

1. Vous rendrez votre travail en exportant votre **projet Java en format .zip**. Préciser le **nom du fichier .zip en y incluant les noms des étudiants de l'équipe**.
2. **Préparez la documentation de votre projet** : l'architecture (au moins le diagramme représentant les modules/composants et classes de votre projet) doit être faite avant de commencer à coder ; à chaque fois que vous avancez sur votre projet, préparez votre javadoc et documentez ce que vous venez de faire et la manière dont vous l'avez faite (dans un fichier partagé entre tous les membres de l'équipe).
3. Faites un rapport avec, au moins :
   a. **Une page de garde** avec l'information de l'équipe de travail

   b. Le **cahier de charges**
   c. Les **nouvelles fonctionnalités que vous avez ajoutées au cahier de charges** de base
   d. L'architecture du logiciel (au moins le **diagramme représentant les modules/composants** et le **diagramme de classes** de votre projet)
   e. **Garanties de qualité telles que les tests, le plan de couverture des tests, des algorithmes d'optimisation** si vous en avez implémentés, comment vous avez réussi à améliorer la généricité et l'extensibilité de vos implémentations, comment vous avez rendu votre projet facile à évoluer…)
   f. **Méthode de travail** (processus suivi, planning du projet, distribution des responsabilités, comment vous avez travaillé ensemble ...)
4. Déposer sous **votre Teams Channel** du cours (si, et seulement si, cette application ne fonctionne pas, veuillez envoyer vos projets dans un email aux enseignants du cours et UN LIEN POUR TÉLÉCHARGER LE PROJET ET LA DOCUMENTATION : surtout, n'incluez pas le projet en tant que pièce jointe de votre courriel, car les filtres de sécurité du serveur SMTP ne laisseront pas passer du code source) le fichier .zip et toute la documentation (ficher .docx ou PDF) de votre projet.
5. Veuillez **respecter le dernier délai pour déposer votre projet** (à la fin de ce document, et aussi dans le slide du planning, voir slides du cours)

Liste de contrôle (checklist) pour L'IMPLEMENTATION de votre projet

1. **Toutes** les **classes, méthodes publiques et champs** de mon projet son **correctement documentées** avec **Javadoc** ?
2. Mon projet a été construit de manière évolutive avec un **Mode Console (CLI)** au début pour la logique et **JavaFX (GUI)** pour la visualisation graphique ?
3. Mon code respecte **la règle DRY** (Do not Repeat Yourself) avec une utilisation efficace de l'**héritage** pour mutualiser les fonctionnalités communes entre les types de classes ?
4. Mon projet a **au moins 10 héritages et 5 interfaces** (où cela soit possible pour l'utilisation actuelle ou pour des éventuelles évolutions de mon programme) ?
5. Mon code respecte **les principes de SOLID** https://medium.com/backticks-tildes/the-s-o-l-i-d-principles-in-pictures-b34ce2f1e898 ? Particulièrement **SRP** (Responsabilité Unique) et **OCP** (Ouvert/Fermé)
6. Mes tests unitaires **couvrent au moins le 90% des fonctionnalités** de mon programme ?
7. Mon code respecte **les conventions de nommage** des variables, constantes, méthodes, et classes ?
8. Mon code traite **les exceptions** de manière adéquate et donner la meilleure résolution à chaque exception soulevée ?
9. Les données sont **correctement encapsulées** ? Avec utilisation correcte des modificateurs d'accès (private, protected), getters/setters avec règles d'accès et modification si nécessaire, et immutabilité (si pertinent).
10. Mon projet **est portable** (toutes les dépendances son sur un gestionnaire de dépendance tel que *Maven ou Gradle,* de manière à ne pas avoir besoin de reconfigurer ou télécharger manuellement quoi que ce soit) ?

Conseils et consignes pour la présentation et démonstration du projet

Vous avez ***20 minutes de présentation et 10 minutes de questions par équipe*** pendant les deux dernières séances du cours pour faire la présentation (en équipe) de vos projets. La note et les commentaires la justifiant vous seront donnés à la fin de chaque présentation/démonstration.

1. **Préparez-vous pour la présentation et démonstration de votre projet** (avez-vous les connecteurs qu'il faut ? avez-vous assez de batterie pour faire la présentation sans interruptions ? vous êtes-vous coordonnés pour l'utilisation de la parole (qui présente quoi) ? …)
2. **Respectez le temps alloué**
3. Pour la partie **présentation**, préparez des slides présentant de manière succincte le cahier des charges, le problème que vous voulez résoudre et en particulier les nouvelles fonctionnalités que vous avez

ajoutées au cahier de charges de base, l'architecture du logiciel (au moins le diagramme représentant les modules et classes de votre projet), des éléments de qualité (si vous avez fait des options donc les algorithmes d'optimisation, de généricité, d'extensibilité, etc…) et des éléments méthodologiques (planning du projet, distribution des responsabilités, comment vous avez travaillé ensemble, refactoring du code...)

4. Pour la **démonstration** il faudra montrer en quoi le programme correspond au cahier de charges en déroulant des scénarios de simulation ; veuillez avoir un esprit critique et objectif de l'outil obtenu, du code produit, et veuillez présenter des perspectives d'évolution.
5. Il vous faudra **répondre aux questions de manière claire, individuelle, concise et avec du recul sur votre projet**.

Projet : Simulateur de Planification et de Gestion d'Actifs Mobiles (SPIGA)

Développer une application orientée objet en Java visant à simuler et à gérer l'opérationnalité d'une flotte hétérogène d'actifs mobiles (aériens, de surface et sous-marins) dans un environnement dynamique.

L'application doit valider les principes de la **Programmation Orientée Objet (POO)** en commençant par une interface en **mode Console (CLI)** pour tester la logique métier, puis en implémentant une interface **Graphique (GUI) en JavaFX** pour la visualisation des opérations.

1. Modélisation de base

1.1. L'Actif Mobile (La Base de la Flotte)

L'actif mobile est l'entité de base de la simulation.

- **Classe Abstraite ActifMobile :** Cette classe doit centraliser les attributs fondamentaux de tous les engins :
    - **Identifiant (ID)**, **Coordonnées 3D (X, Y, Z)** (utilisant des coordonnées flottantes pour la précision), **Vitesse Max**, **Autonomie Max** (en heures ou en pourcentage), et **État Opérationnel** (une énumération : AU_SOL, EN_MISSION, EN_MAINTENANCE, EN_PANNE).
    - La gestion des données doit être rigoureuse (**Encapsulation**), avec des méthodes de modification des coordonnées qui intègrent la gestion des contraintes.
- **Héritage des Milieux :** La flotte sera divisée selon le milieu d'opération :
    - **ActifAerien** (Hérite de ActifMobile) : Drones évoluant principalement en 2D (X, Y) avec une composante Z (Altitude). Ils sont sensibles aux facteurs atmosphériques.
    - **ActifMarin** (Hérite de ActifMobile) : Véhicules de surface (ASV) et sous-marins. Ils sont sensibles aux courants marins et à la profondeur.

1.2. Spécificité des tâches

Les étudiants doivent implémenter au moins 10 classes dans la hiérarchie pour démontrer la maîtrise de l'**héritage** (ActifMobile → VehiculeSousMarin → AUV_Reconnaissance, etc.).

- **Drones aériens :**
    - **DroneReconnaissance :** Spécialisé en vitesse et capacité de surveillance (haute altitude).
    - **DroneLogistique :** Spécialisé en capacité de charge utile et autonomie (vitesse réduite, consommation optimisée).
- **Actifs marins :**
    - **VehiculeSurface :** Opère à la surface (Z constant). Sensible au vent et à l'état de la mer.
    - **VehiculeSousMarin (AUV) :** Opère en 3D sous l'eau. Z est critique (profondeur max/min). Sensible aux courants marins.

1.3. Contrats de comportement

Les comportements transversaux doivent être définis par des **Interfaces** pour respecter les principes SOLID (notamment le Principe de Ségrégation des Interfaces - ISP).

- **Deplacable :** Définit la méthode de base deplacer(cible) et calculerTrajet().
- **Rechargeable :** Définit la méthode recharger() (pour les actifs électriques) ou ravitailler() (pour le carburant).

- **Communicable :** Définit la méthode transmettreAlerte(message, actifCible).
- **Pilotable :** Définit les méthodes demarrer() et eteindre().
- **Alertable :** Définit la méthode notifierEtatCritique(typeAlerte) pour les pannes ou les batteries faibles.

2. Modélisation de l'Environnement et des Contraintes

2.1. La Zone d'Opération (ZoneOperation)

La classe ZoneOperation définit le théâtre des opérations (coordonnées min/max) et doit gérer les facteurs externes dynamiques qui affectent les mouvements de la flotte.

- **Facteurs Atmosphériques et Hydriques :**
    - **Vent :** Modélisé par un **vecteur 2D** (direction et intensité). Il doit augmenter la consommation d'énergie et perturber le cap des **Actifs Aériens** et des **Véhicules de Surface**.
    - **Précipitations :** Valeur d'intensité. Affecte la vitesse maximale et l'autonomie des **Actifs Aériens** (à forte intensité).
    - **Courants Marins :** Modélisé par un **vecteur 3D**. Il doit impacter la vitesse et la consommation d'énergie des **Actifs Marins** (surface et sous-marins).
- **Contraintes Géographiques :**
    - **Obstacles et Zones d'Exclusion :** Collection d'objets Obstacle (ex: une montagne, une zone portuaire). Le mouvement d'un actif doit déclencher une vérification de collision ou d'entrée en zone interdite.

La méthode ActifMobile.deplacer() devra interroger la ZoneOperation pour obtenir les facteurs environnementaux actuels et ajuster le mouvement ou la consommation d'énergie.

3. Gestion de l'Essaim et Planification des Missions

3.1. Gestion des Essaims (Fleet Management)

La classe **GestionnaireEssaim** est le cœur du système. Elle gère la disponibilité des actifs et la composition des groupes.

- **Essaim Hétérogène :** Le gestionnaire doit permettre de créer des **groupes d'intervention temporaires (Essaims)** mélangeant différents types d'actifs (ex: un *DroneReconnaissance* + un *VehiculeSousMarin*).
- **Coordination :** Le gestionnaire doit implémenter une logique de **formation de base** (Pattern Stratégie optionnel) ou d'évitement de collision entre les actifs d'un même essaim.
- **Maintenance :** Le gestionnaire suit l'état des actifs (EN_PANNE, EN_MAINTENANCE) et doit pouvoir suggérer l'actif le plus optimal (selon l'autonomie et l'état) pour une mission entrante.

3.2. Planification des Missions (Mission Planning)

La classe **Mission** doit modéliser une opération complète.

- **Définition de la Mission :**
    - **Objectifs** (énumération : RECONNAISSANCE, SURVEILLANCE, LOGISTIQUE).
    - **Portée** (définition des limites spatiales).
    - **Timeline** (dates/heures de début et fin prévues/réelles).
    - **Résultats Attendus** et **Résultats Obtenus** (pour l'analyse *post-mission*).
- **Association Essaim-Mission :** Une mission est associée à un ou plusieurs actifs.

- **Types de Missions (Héritage) :** Créer des types de missions spécialisées (ex: MissionSurveillanceMaritime, MissionRechercheEtSauvetage).
- **Suivi :** La mission doit avoir un statut (PLANIFIEE, EN_COURS, TERMINEE, ANNULEE). Le système doit permettre de consulter l'historique des missions.

4. Interfaces Utilisateur et Compétences POO

4.1. Interfaces Utilisateur

1. **Interface Console (CLI) :** Permet d'effectuer toutes les opérations de gestion et de simulation (création d'actifs, assignation de missions, affichage des états) avant de passer à l'interface graphique.
2. **Interface Graphique (JavaFX - GUI) :**
    - **Visualisation 2D :** Affichage en temps réel de la ZoneOperation (plan X, Y). Les Actifs doivent être représentés par des icônes dynamiques qui reflètent leur type et leur état.
    - **Tableau de Bord :** Permet la gestion des Essaims et le suivi des missions en cours, ainsi que l'affichage des alertes (pannes, batterie critique).

4.2. Extensions (Points Bonus)

Les étudiants qui veulent aller au-delà des sujets vus en cours peuvent intégrer des mécanismes de POO plus complexes :

- **Optimalisation** : Implémenter une classe OptimaliseurMission qui, selon le Pattern Stratégie, choisit l'actif le mieux adapté à une mission (basé sur l'autonomie, la vitesse et le type d'environnement).
- Intégrer des **algorithmes de *Swarming* avancés** (ex: réorganisation dynamique pour l'évitement de collision, le maintien de formation et la réussite de la mission même en conditions dégradés telles que la perte de drones, le brouillage, un cyber-attaque sur une partie de l'essaim).
- Implémenter la **sérialisation Java** pour la sauvegarde et le chargement complet de l'état de la simulation.
- Intégrer des **graphiques JavaFX** pour la visualisation des données historiques ou de l'autonomie.

Le projet est à rendre en **groupes de 3-4 étudiants**, par le ***Teams Private Chanel* de chaque groupe** en suivant les recommandations et indications pour bien réussir votre projet (données au début de ce document).

Nombre d'étudiants par groupe : 4

Deadline du rendu : dimanche 4 janvier 2026 à minuit.

Soutenances : 7 et 8 janvier 2026.