Guide d'étapes clés : Débuggez une application Java

Comment utiliser ce document ?

Ce guide vous propose un découpage du projet en étapes. Vous pouvez suivre ces étapes selon vos besoins. Dans chacune, vous trouverez :

- des recommandations pour compléter la mission ;
- les points de vigilance à garder en tête ;
- une estimation de votre avancement sur l'ensemble du projet (attention, celui-ci peut varier d'un apprenant à l'autre).

Suivre ce guide vous permettra ainsi :

- d'organiser votre temps ;
- de gagner en autonomie ;
- d'utiliser les cours et ressources de façon efficace ;
- de mobiliser une méthodologie professionnelle que vous pourrez réutiliser.

Gardez en tête que votre progression sur les étapes n'est qu'une estimation, et sera différente selon votre vitesse de progression.

Étape 1 : Mettez en place le repo Git

10 % de progression

Une fois cette étape réalisée :

- vous aurez mis en place le versionning avec Git.

Recommandations :

- Forker le repository https://github.com/OpenClassrooms-Student-Center/Project_DA_Java_EN_Come_to_the_Rescue_of_a_Java_Application/tree/master/Project02Eclipse en un repository public sur votre compte GitHub nommé <nom-prenom-debug-Java>
- Cloner ce nouveau repository sur votre poste de travail en local.

- Créez et pushez une nouvelle branche
    - Créez une nouvelle branche de dev avec `git checkout -b dev`
    - Faites un push avec `git push origin dev`
- Vous travaillerez sur cette branche tout au long du projet, pensez à faire un commit à chaque étape. En fin de projet, il faudra fusionner cette branche sur la branche `main`.

Ressources :

- Cours OpenClassrooms : Gérez du code avec git et github
- Cours OpenClassrooms : Apprenez à utiliser la ligne de commande dans un terminal
- GitHub Learning Lab : tutoriels officiels pour GitHub.

Étape 2 : Analysez le code existant

25 % de progression

Une fois cette étape réalisée :

- vous aurez nettoyé votre code ;
- vous aurez effectué un commit sur le repository.

Recommandations :

- D'abord, lisez et corrigez le code
    - Lisez attentivement le code afin d'identifier ce qui doit être corrigé
    - Exécutez le code avec les commandes suivantes :
        - `javac com/hemebiotech/analytics/AnalyticsCounter.java`
        - `java -cp "." com.hemebiotech.analytics.AnalyticsCounter`
    - Retirez les commentaires inutiles sans altérer le fonctionnement du code existant
- Puis, nous vous conseillons de faire un push des fichiers du projet.
    - Dans le terminal, tapez ou copiez les commandes suivantes :
        - `git add com/hemebiotech/analytics/*.java`
        - `git commit -m 'remove unnecessary comments"`
        - `git push`

Ressources :

- Cours OpenClassrooms : Apprenez à programmer en Java.
- Défi : Gérer un inventaire en Java. Modifiez le code d'une application de gestion d'inventaire, étape par étape.

Étape 3 : Créez une interface pour l'écriture des données

35 % de progression

Une fois cette étape réalisée :

- vous aurez créé l'interface ISymptomWriter et son implémentation WriteSymptomDataToFile.

Pourquoi réaliser cette étape ?

- Pour que le code soit maintenable et évolutif, toute la logique ne doit pas être dans la méthode main d'une seule classe.
    - Les actions doivent être identifiées et isolées dans plusieurs méthodes d'une classe qui sera instanciée.
- Vous allez donc refactorer le code en utilisant des interfaces.
    - En isolant la spécification (l'interface) de l'implémentation (la classe), vous aurez un code plus modulaire : il sera plus facile de modifier l'implémentation d'une classe sans toucher à la manière dont elle est utilisée dans l'application.

Recommandations :

- D'abord, créez votre interface
    - Regardez comment la classe `ISymptomReader` et son interface `ReadSymptomDataFromFile` sont structurées ;
    - Utilisez la même structure en créant :
        - une nouvelle interface `ISymptomWriter` avec une méthode dont le prototype est public void `public void writeSymptoms(Map<String, Integer> symptoms)`;
        - une nouvelle classe `WriteSymptomDataToFile` qui implémente `ISymptomWriter`
        - le corps de la méthode `writeSymptoms` qui écrit les symptômes et leurs quantités dans le fichier result.out au même format que précédemment.
    - Assurez-vous d'avoir bien géré l'exception.
- Puis, faites un push des fichiers du projet
    - Dans le terminal, tapez ou copiez les commandes suivantes :
        - `git add com/hemebiotech/analytics/*.java`
        - `git commit -m "writer implementation"`
        - `git push`

Ressources utiles

- Cours OpenClassrooms : Apprenez à programmer en Java.
- Stackoverflow - Questions Tagged with Java : Utilisez ce site pour poser des questions et trouver des informations utiles
- Learning the Java Language : Documentation Oracle sur des notions clés (contenu rédigé en anglais)

Étape 4 : Refactorez AnalyticsCounter en plusieurs méthodes

50 % de progression

Une fois cette étape réalisée :

- vous aurez découpé toutes les actions menées par AnalyticsCounter en plusieurs méthodes indépendantes.

Recommandations :

- D'abord, refactorez la méthode main de AnalyticsCounter en plusieurs méthodes.
    1. Créez le constructeur avec pour paramètres un objet de type `ISymptomReader` et un objet de type `ISymptomWriter`. Ce constructeur doit assigner les valeurs de ces deux paramètres à deux attributs de classe.
       Prototype : `public AnalyticsCounter(ISymptomReader reader, ISymptomWriter writer) { }`
    2. Créez une méthode `getSymptoms` qui récupère la liste des entrées dans le fichier en utilisant l'instance de `ISymptomReader` déjà créée ;
       Prototype : `public List<String> getSymptoms() { }`
    3. Créez une méthode `countSymptoms` qui compte les occurrences de chaque symptôme existant ;
       Prototype : `public Map<String, Integer> countSymptoms(List<String> symptoms) { }`
    4. Créez une méthode `sortSymptoms` qui trie la liste de symptômes et d'occurrences par ordre alphabétique ;
       Prototype : `public Map<String, Integer> sortSymptoms(Map<String, Integer> symptoms) { }`

5. Créez une méthode `writeSymptoms` qui écrit le résultat dans le fichier de sortie en utilisant l'instance de `ISymptomWriter` déjà créée.
   Prototype : `public void writeSymptoms(Map<String, Integer> symptoms) { }`

- Puis, faites un push des fichiers du projet
  - Dans le terminal, tapez ou copiez les commandes suivantes :
    - `git add com/hemebiotech/analytics/*.java`
    - `git commit -m "refactor analytics counter"`
    - `git push`.

Ressources utiles :

- Cours OpenClassrooms : Apprenez à programmer en Java.
- Stackoverflow - Questions Tagged with Java : Utilisez ce site pour poser des questions et trouver des informations utiles
- Learning the Java Language : Documentation Oracle sur des notions clés (contenu rédigé en anglais)

Étape 5 : Créez la méthode Main

70 % de progression

Une fois cette étape réalisée :

- vous aurez organisé la méthode Main afin de lancer les étapes successives de l'application.

Recommandations :

- Une fois les différents composants de votre application construits, il faut les appeler dans le bon ordre via une méthode qui centralise le déroulé global des étapes
- Créez la classe puis la méthode Main :
  1. Créez une nouvelle classe nommée `Main` ;
  2. Dans cette classe, créez une méthode `main()` qui instancie un objet `ISymptomReader`, un objet `ISymptomWriter` et un objet `AnalyticsCounter` puis qui rappelle les différentes méthodes d'AnalyticsCounter pour exécuter les traitements dans le bon ordre.

- Puis, faites un push des fichiers du projet
    - Dans le terminal, tapez ou copiez les commandes suivantes :
        - `git add com/hemebiotech/analytics/*.java`
        - `git commit -m “main implementation”`
        - `git push`

Ressources utiles :

- Cours OpenClassrooms : Apprenez à programmer en Java.
- Stackoverflow - Questions Tagged with Java : Utilisez ce site pour poser des questions et trouver des informations utiles
- Learning the Java Language : Documentation Oracle sur des notions clés (contenu rédigé en anglais)

Étape 6 : Nettoyez et commentez votre code

100 % de progression

Une fois cette étape réalisée :

- j’aurai terminé mon projet !
- Il ne vous restera plus qu’à justifier mon travail lors de la soutenance.

Recommandations :

- Une fois la classe `AnalyticsCounter` refactorée, il faut s’assurer que le code exécuté tire profit des concepts de la POO et des avantages du langage Java :
    1. Corrigez toutes erreurs dans le naming en appliquant le camelCase ;
    2. Ecrivez la Javadoc de chacune des méthodes ;
    3. Indentez toutes les classes Java et supprimez toutes lignes inutiles.
- Ce projet exige le respect des règles standards listées sur cette page web. Portez votre attention sur le format de la Javadoc et les règles d’indentation.
- Une fois terminé, pensez bien à faire un dernier commit, à fusionner votre branche de travail sur la branche main et à pousser le tout sur le repo en ligne.

Ressources utiles

- Cours OpenClassrooms : Apprenez à programmer en Java.
- Java Naming Conventions (rédigé en anglais).
- Java Map Interface : Tutoriel de JavaTpoint.
- Javadoc Tool : Documentation Oracle sur Javadoc (en anglais).

Projet terminé !