Feuille de TP : Projet Mastermind

1 OBJECTIF DU PROJET

Durant ce projet, vous implémenterez les classes permettant de jouer au jeu Mastermind.
Mastermind est un jeu dans lequel un joueur a un certain nombre de tentatives pour deviner un code couleur.

- Dans un premier temps, vous coderez les classes permettant de stocker les données du jeu.
- Dans un second temps, vous compléterez la classe **Game**, permettant de faire une partie sans interface graphique
- Enfin, vous terminerez votre projet par la finalisation de la classe **GUI**, permettant de jouer au jeu avec une interface graphique

LISEZ BIEN LE SUJET ET AVANCEZ ETAPE PAR ETAPE.

2 ENONCE

2.1 PRESENTATION DU JEU

Dans cette version du jeu mastermind, un joueur tente de découvrir un code composé de quatre cases, chacune contenant une couleur. Une couleur peut être présente plusieurs fois dans un code.

Exemple :

Figure 1- Exemple de code à deviner

Ici, le code que dois deviner le joueur : ROUGE | MAGENTA | MAGENTA |CYAN

Le joueur a donc 10 tentatives pour deviner ce code. A chaque tentative :

1. Le joueur fait une proposition de code :

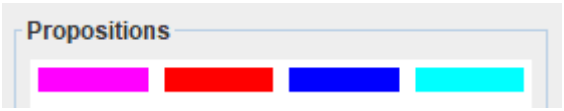


Figure 2 - Proposition du joueur

Ici le joueur propose le code suivant : MAGENTA | ROUGE | BLEU | CYAN

2. Le logiciel donne des indications sur la proposition du joueur :
   Pour chaque case, le logiciel renvoi :
   i. VERT si le joueur a proposé la couleur correcte
   ii. NOIR si le joueur a proposé une couleur présente dans le code, mais mal placée
   iii. BLANC si la couleur n'est pas présente dans le code

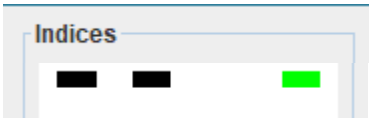


Figure 3 - Indications en réponse à la proposition du joueur

Ici, le logiciel répond :

a) Pour la première case : NOIR, car le joueur a proposé MAGENTA. Cette couleur est présente dans le code, mais pas dans cette case (elle est présente dans la case 2).
b) Pour la deuxième case : NOIR, car le joueur a proposé ROUGE. Cette couleur est présente dans le code, mais pas dans cette case (elle est présente dans la case 1).
c) Pour la troisième case : BLANC, car la couleur proposée par le joueur (BLEU) n'est pas présente dans le code à deviner.
d) Pour la quatrième case : VERT, car le joueur a proposé CYAN, ce qui est la bonne couleur pour cette case.

Ensuite, le joueur peut faire une nouvelle proposition, en tenant compte des indices fournies par le logiciel.

Le jeu s'arrête lorsque le joueur fait une proposition identique au code à deviner, ou lorsque le joueur a fait 10 propositions sans succès.

Pour une vidéo de démonstration :

https://www.youtube.com/watch?v=NVDLhpGdHgY

2.2 SUJET

Afin de réaliser au mieux votre projet, veillez à créer un projet JAVA à part sur eclipse, qui ne contiendra que les fichiers de votre projet. (File/New/Java Project)

Le programme de ce logiciel se décompose en quatre packages que vous devez créer :

1. data_structure : contiendra les classes relatives aux jetons et aux lignes
2. graphics : contient les classes de l'interface graphique que VOUS NE DEVEZ PAS MODIFIER
3. main : contient les classes à exécuter pour jouer (en mode graphique et non graphique)
4. test : contient des classes pour tester vos codes que VOUS NE DEVEZ PAS MODIFIER

Vous pouvez d'ores et déjà créer ces quatre packages, et récuperer sur la plateforme les classes des packages graphics et test.

2.2.1 Etape 1 : Les jetons

Nous allons d'abord coder des classes permettant de représenter une couleur dans une case, nous appellerons ces objets des jetons. Nous avons besoins de deux types de jetons, qui ont des propriétés communes.

a) Combien de classes sont nécessaires ?
b) Proposez une hiérarchie de classe

Nous allons concevoir une première classe de jeton globale. Cette classe ne pourra pas être instanciée.

c) Récuperer la classe Jeton.java sur la plateforme. Lisez attentivement son code.
    1. Rendez la classe impossible à instancier.
    2. Ajouter un attribut public de type String nommé **_color** (n'oubliez pas le « _ »)**.**
    3. Ajouter un constructeur vide qui initialise l'attribut _color à « NONE ».
    4. Ajouter une méthode de comparaison **equals**. Cette méthode prend en argument un jeton, et renvoi **true** si les jetons ont des attributs _color égaux. Elle renvoie **false** sinon.
d) Nous allons maintenant ajouter une classe **Jeton_indice** dont les valeurs ne peuvent être que « VERT | NOIR | BLANC ». Ces jetons servent pour donner les indices au joueur.
    1. Récuperer la classe Jeton_indice.java sur la plateforme. Lisez attentivement son code.
    2. Ajouter l'héritage nécessaire.
    3. Ajouter un constructeur prenant en argument un entier n, et initialisant l'attribut _color de l'objet avec la valeur de la case d'indice n du tableau **USED_COLORS**.

Vous pouvez (et devez) désormais tester vos classes Jeton et Jeton_indice en exécutant la classe **Test_Jeton_Jeton_indice**. Si aucun message en rouge ne s'affiche entre le début et la fin du test, votre code a de fortes chances d'être correcte.

e) Pour finir cette fonctionnalité, nous allons maintenant coder la classe **Jeton_couleur**, dont les valeurs peuvent être « RED | BLUE | CYAN | YELLOW | MAGENTA | ORANGE ». Ces jetons serviront pour constituer le code à deviner ainsi que les propositions du joueur.
    1. Récuperer la classe Jeton_couleur.java sur la plateforme. Lisez attentivement son code.
    2. Ajouter l'héritage nécessaire.
    3. Ajouter un constructeur vide qui initialise l'attribut _color du jeton avec une case aléatoire du tableau USED_COLORS. Pour générer aléatoirement un int, vous pouvez utiliser une instance de la classe **Random r**. Depuis cette classe vous pouvez appeler la fonction nextInt(int i), qui renvoi un entier compris entre 0 et i-1.
    4. Ajouter un autre constructeur prenant en entrée un entier i, et initialisant l'attribut _color avec la valeur de la case d'indice i du tableau USED_COLORS.
    5. Ajouter un getter **getUSED_COLORS** renvoyant l'attribut **USED_COLORS.**
    6. Ajouter un getter **getColors** renvoyant l'attribut **colors.**

    NB : Pour les questions 5. Et 6., vérifiez bien la nature des attributs que vous souhaitez renvoyer.

Vous devrez compléter la méthode **evaluation** dans l'étape 2.

2.2.2 Etape 2 : Les lignes

Nous allons maintenant coder la classe **Row**. Une classe générique permettant de stocker des jetons.

a) Récuperer la classe Row.java sur la plateforme. Lisez attentivement son code.
b) Rendez la classe Row générique, qui n'accepte comme classe paramètre que des classes filles de la classe Jeton.
c) Coder la méthode **equals** qui renvoie « true » si la ligne passée en paramètre est égale à la ligne appelante (this). Plus précisément, si les deux lignes contiennent des jetons de valeurs égales dans chaque case. Renvoie false sinon.

   NB : Utilisez les méthodes écrites dans les classes jetons.

d) Coder la méthode **toString** pour qu'elle renvoie un string contenant les valeurs de chaque case, séparées par des « | ».

   Exemple : « | RED | BLUE | CYAN | CYAN |»

   NB : Utilisez la méthode **toStringCentered** de la classe Jeton.

Les prochaines méthodes servent à évaluer la proposition du joueur et à savoir s'il a trouvé le bon code.

e) Allez dans la classe **Jeton_couleur** pour compléter la méthode **evaluation.** Cette méthode évalue si le Jeton_couleur (this) correspond à la valeur du jeton à la position **pos** de la Row **solutionGagnante**. Cette méthode prend deux paramètres :

   - Row solutionGagnante : la solution à laquelle on compare la proposition
   - int pos : l'indice de la case de solutionGagnante contenant le jeton devant être comparé au jeton this.
   -

   Cette méthode compare donc deux jetons, et renvoie un jeton de couleur :
   - GREEN (0) si le jeton (this) à la même valeur que le jeton de solutionGagnante à la position pos
   - BLACK (1) si le jeton (this) à la même valeur que n'importe quel autre jeton indice de solutionGagnante
   - WHITE (2) sinon

Vous pouvez (et devez) désormais tester votre classe Jeton_couleur en exécutant la classe **Test_Jeton_Couleur**. Si aucun message en rouge ne s'affiche entre le début et la fin du test, votre code a de fortes chances d'être correcte.

f) Retournez dans la classe **Row** pour compléter la méthode **computeSoluce**. Cette méthode sera appelée depuis la ligne gagnante. Elle renvoie un objet de type Row, contenant des jetons_indices correspondant à l'évaluation de la proposition passée en arguments, par rapport à la Row (this) du code à deviner.

   Exemple :

   Si computeSoluce est appelée depuis la ligne gagnante contenant les valeurs :

```
|  RED  | BLUE  |  RED  |  RED  |
```

avec en paramètre la proposition contenant les valeurs:

```
|  RED  |  RED  | BLUE  |YELLOW |
```

la méthode renverra une Row contenant les valeurs :

```
| GREEN | BLACK | BLACK | WHITE |
```

NB : Utilisez la méthode evaluation de la classe Jeton_couleur codée dans la question précédente.

g) Complétez la méthode **succeed**. Cette méthode renvoie **true** si la Row (this) ne contient que des Jeton_indice de valeur «GREEN». Elle renvoie **false** sinon.

NB : Utilisez la méthode equals de votre classe Jeton

Vous pouvez (et devez) désormais tester votre classe Row en exécutant la classe **Test_Row**. Si aucun message en rouge ne s'affiche entre le début et la fin du test, votre code a de fortes chances d'être correcte.

2.2.3 Etape 3 : Jeu sans interface graphique

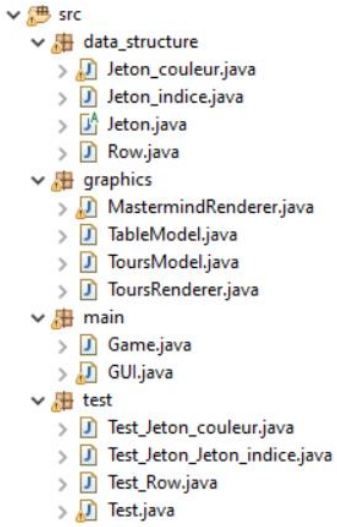
Nous allons maintenant coder l'algorithme réalisant les tours de jeu. Pour ce faire, nous allons mettre au propre l'arborescence de notre programme.

a) Si ce n'est pas fait, créer un package **data_structure** et mettez vos classes Jeton, Jeton_indice, Jeton_couleur, Row.

b) Si ce n'est pas fait, créer trois packages, **graphics** et **main** et **test**. Récuperer sur la plateforme les classes et mettez-les dans les packages comme suit :

- graphics :
  - MastermindRenderer.java
  - TableModel.java
  - ToursModel.java
  - ToursRenderer.java
- main :
  - Game.java
  - GUI.java
- Test
  - Test_Jeton_couleur
  - Test_Jeton_Jeton_Indice
  - Test_Row
  - Test

Votre arborescence doit donc ressembler à :



Nous allons maintenant pouvoir coder.

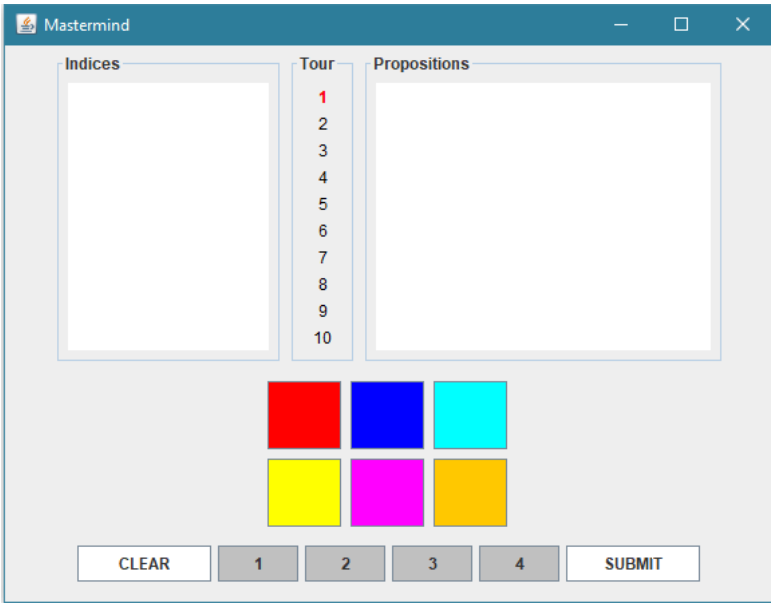
c) Commencez par réfléchir aux différentes étapes d'un tour de jeu. Ecrivez ensuite votre algorithme en pseudo-code permettant de réaliser une partie de Mastermind complète, et faite le vérifiez par votre enseignant de TP.
d) Lisez attentivement le code de la classe **Game**, et prenez note des fonctions qui pourront vous être utiles pour réaliser une partie de Mastermind.
e) Complétez la méthode **nonGraphicsGame** de la classe **Game** en vous appuyant sur votre pseudo-code et en suivant étape par étape les commentaires de la méthode.

   NB : Appuyez-vous sur les méthodes déjà présentes dans la classe Game et sur les méthodes que vous avez codé précédemment.

Vous pouvez désormais exécuter la classe **Game** pour tester si votre code fonctionne. Si vous arrivez à faire une partie, vous avez réussi, bravo !

2.2.4 Etape 4 : Jeu avec interface graphique

Nous allons maintenant compléter le code permettant de jouer avec une interface graphique comme suit :

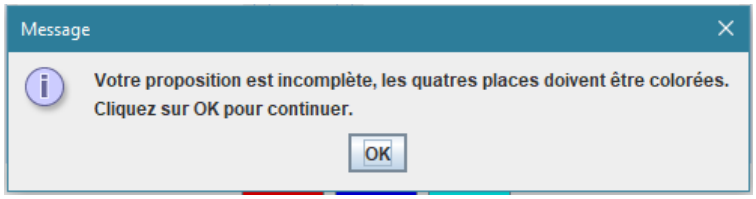


a) Commençons par compléter dans la classe **Game** la méthode **evalProposition**. Cette méthode doit
   - Récuperez dans une variable l'évaluation de l'attribut **proposition** par la ligne **ligneGagnante.**
   - Stockez dans le tableau **responses_data.donnees** à la ligne numéro **counter**, la proposition convertie en tableau (utilisez une méthode de la classe Row).
   - Stockez dans le tableau **indicators_data.donnees** à la ligne numéro **counter**, l'évaluation de la proposition convertie en tableau (utilisez une méthode de la classe Row).
   - Mettez à jour l'interface graphique en appelant depuis **responses_data** et **indicators_data** la méthode **fireTableDataChanged()**.
   - Incrémentez le compteur de tour (attribut de la classe Game).
   - Renvoyez true si la proposition est correcte (égale au code), false sinon. Pour cela, utilisez simplement la méthode succeed.

b) Pour terminer ce programme, complétez la méthode **actionPerformed** de la classe **GUI**. Cette méthode est appelée lorsqu'un bouton est cliqué. On gère quel bouton a été cliqué avec des if. Suivez les commentaires pour compléter la partie manquante de la fonction (le comportement à adopter lorsque le bouton SUBMIT est cliqué, c'est-à-dire lorsque le joueur fait une proposition).

Pour afficher un message sur l'interface graphique, vous pouvez utiliser :

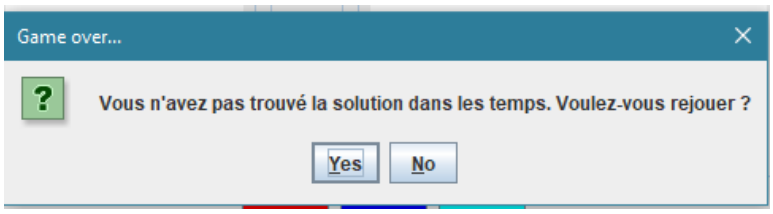
```
JOptionPane.showMessageDialog(mainFrame, "Votre proposition est incomplète, les quatres places doivent être colorées. \nCliquez sur OK pour continuer.");
```



Modifiez le deuxième paramètre de la méthode pour personnaliser le message comme vous le souhaitez.

Pour afficher un message avec une question fermée, vous pouvez utiliser :

```
JOptionPane.showConfirmDialog(mainFrame," Vous avez trouvé la solution en"+theGame.getCounter()+" coups. Voulez-vous rejouer ?","Félicitation !",JOptionPane.YES_NO_OPTION);
```



Modifiez le deuxième paramètre de la méthode pour personnaliser le message comme vous le souhaitez. N'oubliez pas de récuperer le retour de la fonction !

NB : La classe GUI dispose d'un attribut de la classe **Game** nommé **theGame**. Cette classe contient toutes les données du jeu. N'hésitez pas à utiliser les méthodes de la classe Game à partir de cet attribut dans votre algorithme.

Vous pouvez désormais lancer votre partie en exécutant la classe **GUI**.

2.2.5 Etape bonus : fonctionnalités complémentaires

Si vous avez fini les parties précédentes du sujet, vous pouvez vous attaquer aux problématiques suivantes (dans l'ordre que vous voulez):

a) JOUEUR :
   - Créez une interface Joueur qui permet de faire jouer soit un Joueur Humain, soit un Bot (un joueur joué par l'ordinateur)
   - créer deux classes (qui implémentent l'interface Joueur) :
     - bot bête (qui joue aléatoirement)
     - bot intelligent (qui essaye de gagner)
   - Adaptez votre code pour pouvoir faire jouer un bot sans interface graphique (grâce à l'interface Joueur que vous avez écrit)
   - Adaptez votre code pour pouvoir faire jouer un bot avec interface graphique

b) SAUVEGARDE
   - Adaptez votre code pour permettre de sauvegarder une partie en cours de jeu
     - Vous pouvez pour cela effectuer une action lorsque l'utilisateur ferme la fenêtre. Cherchez sur internet la méthode qui est appelée lorsque le joueur ferme la fenêtre, et surchargeait cette méthode.
     - Pour la sauvegarde, vous pouvez simplement créer un fichier et écrire dedans les informations pertinentes.
   - Adaptez votre code pour permettre de charger une partie sauvegardée.

3 CONDITIONS DE REPONSE

Vous devrez rendre votre projet pour le 14/12/2022 - 23h59.

Veillez à bien suivre les instructions décrites ci-dessous, sous peine d'avoir 0/20.

Cette deadline est FIXE, tout retard entrainera une perte de 1 points par heure. Il n'est donc plus nécessaire de rendre votre projet après 20h le 15/12/2022.

3.1 PARTICIPATION INDIVIDUELLE

Ce projet a été conçu pour être réalisée de manière individuelle. Tout travail groupé et toute copie, même partielle du code d'un camarade sera sanctionnée par un 0/20, quelle que soit la justification.

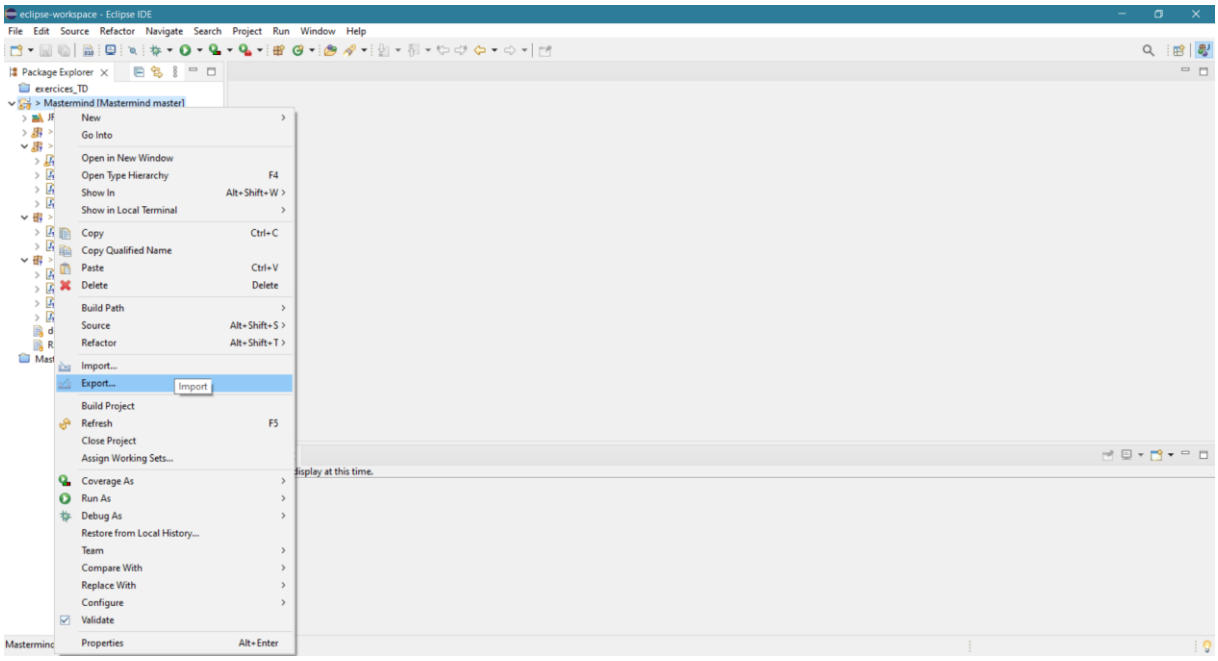
Il ne suffit pas de changer le nom d'une variable pour éviter d'être détecté comme copie, vous devez produire votre propre algorithme. Ceci fera l'objet d'une attention particulière.

Donnez le meilleur de vous-même, avec l'aide de l'enseignant, en étant impliqué, vous vous assurerez la moyenne sans soucis.

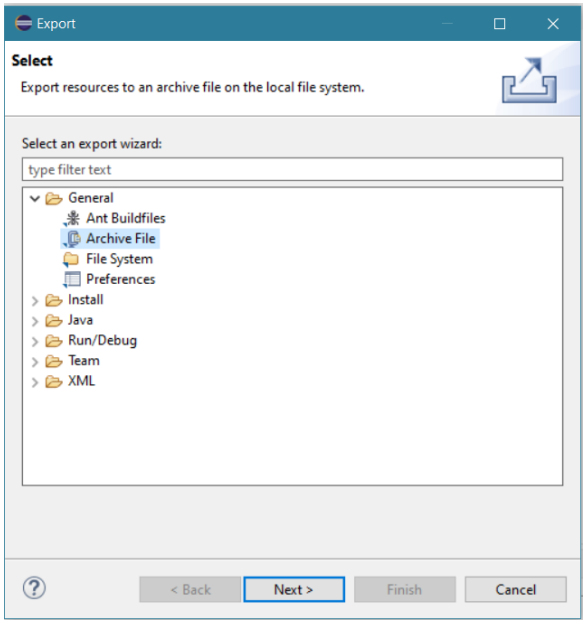
3.2 CREER VOTRE ARCHIVE

Seuls les dépôts au format valide seront acceptés. Vous devez déposer une archive complète de votre projet. Pour ce faire, suivez les instructions suivantes depuis l'interface eclipse, avec seulement votre projet ouvert.

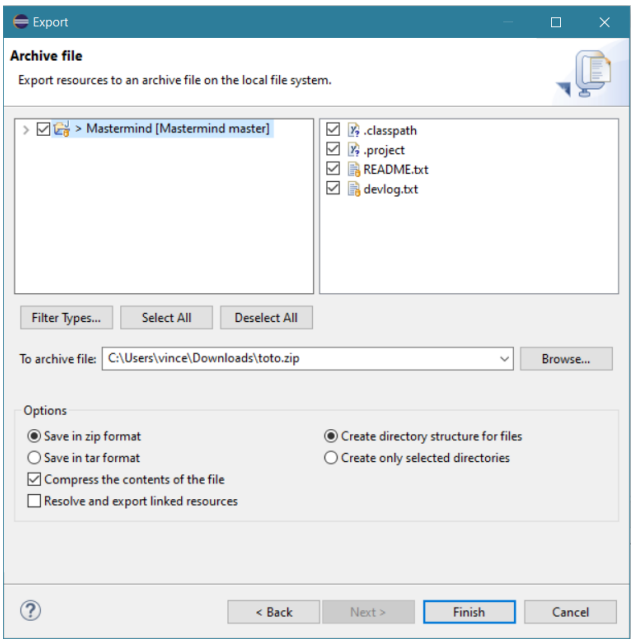
a) Clique droit sur votre projet, puis exporter :



b) General/Archive File



c) Veillez à ce que votre projet soit coché, ainsi que .classpath et .project.
d) Indiquez dans « To archive file » l'endroit ou vous allez sauvegarder votre archive et son nom
e) Veillez à ce que l'archive soit au format .zip
f) Cliquez sur finish.



g) Votre archive a été créée ! Veillez à bien la déposer sur la plateforme avant la date limite.