TP 1 – Eclipse, Jeu de Bataille

Objectif(s)

- ✯ être capable de créer un projet java dans Eclipse
- ✯ être capable d'accéder et d'exploiter des classes existantes
- ✯ être capable de créer, de modifier des classes dans un projet

Préparer Eclipse

Exercice 1 – Où sont les fichiers de mon projet

Il s'agit de préparer l'environnement de l'IDE Eclipse pour le développement d'un projet Java.

1. Tous les fichiers de vos projets pour la session courante sont stockés dans un dossier spécifique. Les caractéristiques de ce dossier sont accessibles dans les `Preferences` d'Eclipse. Saurez-vous dire quel est le chemin et l'encodage de ce dossier ?
2. La dernière version Java SE LTS (Long-Term-Support) est la 17, mais nous pouvons aussi travailler avec la version 11. Quelles sont celles disponibles sur votre machine ? Si la 11 (ou ultérieur) n'est pas disponible, procéder à une mise à jour.
3. Créer un projet Java nommé `tp1`. Dans quel dossier se trouveront vos fichiers compilés (bytecode) ?
   — Quand Eclipse vous demande de créer un `module-info.java`, crèez-le.
4. Créer une classe `Test` contenant une méthode `main` en cochant la case spécifique et pour un package nommé `exercices`. Dans quel dossier se trouve votre fichier et quel est son nom ?

Simple classes/méthodes

Exercice 2 – Maintenant on code

Question 1

Compléter votre classe `Test` en ajoutant la méthode :

```
public void afficherMessage(String message) {
    System.out.println(message);
}
```

puis en complétant `main` :

```
Test t = new Test();
t.afficherMessage("échec");
```

Question 2

L'exécution du code affiche "échec" dans la partie `console` de l'interface. Sauriez-vous exécuter ce code dans le terminal sans Eclipse ?

Modifier le code pour qu'il affiche "succès".

Exercice 3 – Avec deux classes

Il s'agit d'isoler la partie ré-utilisable du code, de l'exécution sur un cas concret : un nombre entier est-il positif ?

Question 1

Renommer (menu `refactor`) la classe en `TestPositif`.

Question 2

Remplacer la méthode `main` par :

```
public void testerPositif(int n) {
    if(0 <= n) {
        this.afficherMessage("succès");
    } else {
        this.afficherMessage("échec");
    }
}
```

Question 3

Créer puis tester une classe `Main` dans le même package dont l'unique méthode est `main` :

```
public static void main(String[] args) {
    TestPositif t = new TestPositif();
    t.testerPositif(10);
    t.testerPositif(-10);
}
```

Question 4

Nous allons numéroter les messages à l'aide d'un compteur lié à la classe `TestPositif` en ajoutant :

```
public static int numeroTest = 1;
```

1. Modifier la méthode `afficherMessage` :

   ```
   System.out.println(numeroTest + " : " + message);
   ```

2. Où doit-on incrémenter le compteur ?
3. Remplacer l'usage du `println` par la méthode `printf` qui se comporte comme la fonction du C.

Question 5

Compiler et exécuter votre projet sans passer par Eclipse.

Le jeu de bataille

Exercice 4 – Importer un .JAR dans le *build-path*

Pour un projet, Java utilise depuis sa version 9 deux chemins séparés : le `class-path` et le `module-path`. Le premier sert pour accéder à vos classes, le second aux modules que vous utilisez.

Ajoutez à votre projet le module `Cartes.jar`. Pour cela, vous devez :

— Sélectionnez votre projet dans l'explorateur de *packages*.

— Accéder au *build-path* de votre projet :
Solution 1 : avec le menu

  — Ouvrez le menu *Project*

  — Choisissez l'élément *Properties*

  — Choisissez la catégorie (à gauche dans le dialogue) *Java Build Path*

Solution 2 : avec la souris

  — Clic-droit sur le projet

  — Élément *Build Path*

  — Sous-élément *Configure Build Path...*

— Dans le dialogue qui s'affiche, vous avez 5 onglets :

Source la définition de votre (vos) dossier(s) contenant vos fichiers `Java`, et le dossier qui recevra les fichiers `class` compilés,

Projects la définition des projets (dans votre *workspace*) dont votre projet a besoin,

Libraries la définition des bibliothèques et des modules dont votre projet a besoin,

Order and Export la définition de ce que votre projet propose aux autres projets eclipse,

Module Dependencies la définition des dépendances des modules.

— Sélectionnez l'onglet *Libraries*

— Dans la partie centrale se trouve deux entrées : le `module-path` et le `class-path`. Sélectionnez le premier (qui contient déjà le JRE).

— Cliquez sur le bouton *Add External JARs...* (à droite du dialogue).

— Allez chercher votre .JAR à l'endroit où vous l'avez téléchargé et cliquez sur le bouton Ouvrir.

— Optionnellement, vous pouvez attacher la javadoc du module au .JAR.

— Fermez le dialogue. Le module doit apparaître dans une section *Referenced Libraries* du *Package Explorer*.

— Ouvrez le fichier `module-info.java`.

— Ajoutez entre les accolades une ligne "**requires** Cartes ;".

Exercice 5 – Apprivoiser le module Cartes

Question 1

Ajoutez à votre projet le fichier `TestCartes.java`, dans le package `bataille`.
Compilez votre projet et testez le.

La fonction `main` crée un objet de la classe **TestCartes** (ce qui exige appeler le constructeur de cette classe) et appelle sa méthode `testeCartes()`.

La classe **TestCartes** contient un attribut de type **PaquetCartes**, lequel est instancié dans le constructeur.
La méthode `testeCartes()` crée trois objets de type **Carte** et les ajoute au paquet.

Des affichages permettent de visualiser les objets créés. La méthode `toString()` de la classe **PaquetCartes** est définie, vous pouvez donc directement passer une référence vers un objet de cette classe à `println()`.

Prenez le temps pour regarder la JavaDoc de **PaquetCartes** et prendre compte de ses méthodes.

La JavaDoc des différentes classes est téléchargeables en forme du fichier `bataille_doc.zip`.

Question 2 32 cartes sinon rien...

Ajoutez à la classe la méthode `remplitPaquet` qui va créer les 32 cartes d'un jeu usuel et qui les ajoutera au paquet, que vous afficherez.

Appelez votre méthode depuis la fonction `main` et testez !

Question 3 Battu c'est encore mieux!

La classe **PaquetCartes** contient une méthode `melanger`. Utilisez donc cette méthode pour mélanger le paquet de cartes avant de l'afficher.

Question 4 Tirer des cartes.

Ajoutez une nouvelle méthode `testeTirer` à votre classe. Dans cette méthode, vous tirerez les cartes du paquet une à une avant de les afficher. Il vous faudra donc une boucle ***while***.

Testez votre programme.

Exercice 6 – La bataille

Question 1 A vous de jouer

Récupérez le fichier **JeuBataille.java** qui contient un squelette que vous allez devoir compléter. La classe **JeuBataille.java** a deux attributs : `joueur1` et `joueur2`. Profitez pour aller voir dans la Javadoc les méthodes de la classe `Joueur`.

Question 2 Les joueurs et les cartes

Dans le constructeur de la classe **JeuBataille.java** qui prend 2 noms de joueurs en argument, il faut :

— créer les deux joueurs

— créer un paquet de cartes vide

— créer les 32 cartes d'un jeu et les ajouter au paquet

Vous pouvez tester le bon fonctionnement de votre constructeur en affichant les joueurs et le paquet de cartes dans la méthode `aJouer`.

Question 3 Mélange et distribution des cartes

Dans la méthode `jouer` de la classe **JeuBataille.java**, il faut maintenant mélanger les cartes du paquet et distribuer les cartes une par une entre les deux joueurs. Pour la distribution, il faudra faire une boucle qui prendra les cartes une par une au dessus du paquet de cartes et la donnera alternativement à chaque joueur qui la mettra en dessous de son paquet.

Remarque:

On va commencer par une version simplifiée du jeu de la bataille. Chaque joueur tire une carte du dessus de son paquet. Le joueur qui a la carte la plus haute remporte les deux cartes qu'il place en dessous de son paquet. En cas d'égalité chaque joueur place sa carte en dessous de son paquet

Question 4 On joue 1 tour

Écrire une nouvelle méthode privée appelée `jouer1tour` et qui renvoie une référence sur un joueur. La méthode :

— vérifie que le `joueur1` a au moins une carte dans son paquet. Si ce n'est pas le cas, le `joueur1` ne peut pas jouer. Il a perdu et la méthode retourne `joueur2`.

— vérifie que le `joueur2` a au moins une carte dans son paquet. Si ce n'est pas le cas, le `joueur2` ne peut pas jouer. Il a perdu et la méthode retourne `joueur1`.

— implante la version simplifiée du jeu.

— se termine et retourne la valeur `null` car personne ne peut être déclaré vainqueur

Question 5 On fait 100 tours de jeu ?

Dans la methode `jouer`, faire une boucle qui fait 100 itérations de jeu sauf si un joueur gagne avant. Si au bout de 100 tours les deux joueurs ont encore des cartes, on compte le nombre de cartes de chaque joueur et celui qui en a le plus est déclaré vainqueur, sinon, il y a égalité entre les deux joueurs.

Question 6 Deuxième version

Chaque joueur tire une carte du dessus de son paquet. Le joueur qui a la carte la plus haute remporte les deux cartes qu'il place en dessous de son paquet. En cas d'égalité, chaque joueur tire deux nouvelles cartes. Les premières cartes du 2e tirage de chaque joueur sont comparées. Le joueur qui a la carte la plus haute remporte les six cartes. En cas d'égalité, chaque joueur remet ses 3 cartes dans son paquet.

Vous devez implanter cette nouvelle règle dans la méthode `jouer1tour`. Vous prendrez soin de vérifier que les paquets de cartes des joueurs ont suffisamment de cartes avant de les tirer.