Yann.Secq@univ-lille.fr
Fabien.Delecroix@univ-lille.fr

Objectifs: Réfléchir au projet et pratiquer les types et la décomposition en fonctions.

1 SAÉ : Logiciels Ludo-pédagogiques

Demander un avis sur votre descriptif de projet à votre client.e.

2 Création et utilisation de types : mise en pratique

Exercice 1 : Retour sur le jeu du pendu [V]

ous allez refaire un programme qui permet de jouer au jeu du pendu, mais cette fois vous allez décomposer le problème en sous-fonctions, et définir deux nouveaux type pour représenter les données manipulées.

Le joueur doit découvrir un mot caché. Au début, toutes les lettres du mot sont cachées et représentée à l'écran par des étoiles, indiquant la longueur du mot à découvrir. À chaque tour du jeu le joueur propose une lettre de l'alphabet. Si la lettre est présente dans le mot, elle est découverte. Sinon, le joueur perd un essai. Généralement le joueur a droit à 5 essais pour découvrir la totalité du mot.

2.1 Choix de la structure de données et définition des nouveaux types

Dans le jeu du pendu, nous devons manipuler des lettres et des mots. Ce sont donc deux types assez naturels à définir.

Le type `Lettre` est défini par un caractère `car`, et un booléen `decouverte` qui indique si la lettre est découverte ou non.

Rappelons que chaque nouveau type doit être défini dans un fichier séparé.

Q1. Créer le répertoire `pendu` qui contiendra tous les fichiers pour la résolution du jeu du pendu.

Q2. Définir le type `Lettre` dans un fichier `Lettre.java`, et qui contient les champs décrits ci-dessus.

Q3. Créer le fichier `Pendu.java` qui contiendra l'algorithme principal. Dans ce fichier, écrire la fonction `Lettre newLettre(char car)` qui crée une lettre non découverte.

Q4. Écrire la fonction de test `void testNewLettre` qui contiendra 2 assertions. Chaque assertion va créer une lettre, la stocker dans une variable, puis tester que le champ `car` de cette lettre est bien celui donné à la création, et que le champ `decouvert` est `faux`. Exécutez le test.

Q5. Dans le fichier `Pendu.java`, écrire la fonction `Lettre[] creerMot(String mot)` qui crée un tableau qui contient les `Lettres` de la chaine `mot`. Pensez à utiliser `newLettre`.

Q6. Écrire la fonction de test `void testCreerMot()` qui teste que la fonction `creerMot` retourne le tableau de lettres comme attendu. Pour cela, vous allez déclarer une variable `mot` de type `Lettre[]` initialisée avec `"unix"`, puis tester que `mot` a une longueur de 4, que `mot[0].car` est caractère `'u'`, et que `mot[2].car` est le caractère `'i'`.

2.2 Décomposition du problème

Maintenant que les structures de données sont définies, on peut s'intéresser à la décomposition du problème.

Commencez par écrire (sur papier, ou en commentaire dans votre fichier) une ébauche de `void algorithm()` qui contiendra l'initialisation du mot secret, la boucle principale avec ses différentes étapes, puis l'affichage du résultat final. Ceci vous permet de déterminer les fonctions qu'il est nécessaire de définir.

Il y a plusieurs décompositions possibles, qui demanderont d'écrire des fonctions différentes. **Dans tous les cas, chacune de ces fonctions doit être testée. De plus, le test de chaque fonction sera écrit immédiatement avant ou après d'écrire le corps de la fonction.** Les fonctions de saisie sont testées à l'aide de `void algorithm()`, toutes les autres fonctions sont testées avec des fonctions de test dédiées.

Dans la suite vous serez guidés pour écrire les fonctions qui correspondent à une certaine décomposition. Vous pouvez vous en inspirer et éventuellement les adapter à votre proposition de décomposition.

Q7. Écrire la fonction `String toString (Lettre lettre)` qui retourne la chaîne `"*"` si `lettre` n'est pas découverte, et la chaîne contenant le caractère correspondant à la lettre sinon.

Q8. Écrire la fonction `void testToStringLettre()` qui contient au moins deux assertions permettant de tester que le résultat retourné par `toString` est correct. Ces assertions devront tester les deux cas: un pour une lettre non découverte, et un pour une lettre découverte.

Q9. Écrire la fonction `String toString (Lettre[] mot)` qui affiche la représentation d'un mot sous forme de chaîne de caractères. Vous devez réutiliser la fonction `String toString(Lettre lettre)`.

Q10. Écrire `void testToStringMot()` approprié.

Q11. Écrire la fonction `boolean estDecouvert(Lettre[] mot)` qui permet de déterminer si le mot donné en paramètre est entièrement découvert.

Q12. Écrire le test `void testEstDecouvert()`. Dans le test vous devez avoir au moins trois assertions correspondant à ces cas:

- mot entièrement découvert
- mot non entièrement découvert
- mot de longueur 0

Q13. Écrire la fonction `boolean decouvrir(Lettre[] mot, char car)` qui découvre toutes les occurrences de la lettre `car` dans `mot`. La fonction retourne vrai si la lettre appartient au mot (càd au moins une occurrence a été découvrte), et faux sinon.

Q14. Écrire la fonction de test `void testDecouvrir()` pour la fonction de la question précédente.

Q15. Finalement, écrire la fonction `void algorithm()` et tester votre implémentation du pendu.

Exercice 2 : Gestion de parc de véhicules

On s'intéresse dans cet exercice à la gestion (très simplifiée) d'un parc de véhicules.

Q1 : Représentation des véhicules

Un véhicule est caractérisé par un modèle (de type String) et une année (de type int). Définissez le type Vehicule constitué de deux champs pour stocker son modèle (champ nommé modele) et son année de construction (champ nommé annee).

Q2 : Constructeur pour un véhicule

Ecrire la fonction Vehicule `newVehicule (String modele, int annee)` qui permet de construire un véhicule.

Q3 : Représentation d'un véhicule

Ecrire la fonction `String toString (Vehicule v)` qui crée une chaîne de caractères permettant de représenter un véhicule. La fonction devra valider le test suivant :

```
void testToStringVehicle(){
  assertEquals("Yaris 2020", toString(newVehicule("Yaris", 2020)));
}
```

Q4 :Représentation d'un parc de véhicules

Considérons un parc de véhicules. Définissez le type `Parc` constitué de deux champs :

- vehicules qui est un tableau à une dimension de véhicules ;
- nbVehicules de type entier qui permet de stocker le nombre effectif de véhicules dans le parc.

Q5 : Constructeur pour un parc de véhicules

Écrire la fonction `Parc newParcVide()` qui construit un parc de véhicules pouvant contenir le nombre maximal de véhicules, mais qui est pour l'instant vide. Remarque : on supposera que la constante est déjà définie.

```
final int NB_MAX=100;
```

Cette constante correspond au nombre maximum de véhicules qu'un parc peut contenir.

Q6 : Ajout d'un véhicule

Écrire la fonction `boolean ajouterVehicule (Parc parking, Vehicule v)` qui ajoute un nouveau véhicule au parc de véhicules. Si le parking est déjà plein, cette fonction n'ajoute pas le véhicule, laisse donc le parc inchangé et retourne la valeur `false`. A l'inverse, si le véhicule a été ajouté, la fonction retourne `true`.

Q7 : Représentation d'un parc de véhicules

Écrire la fonction `String toString (Parc parking)` qui crée une chaîne de caractères permettant de représenter un parc de véhicules. La fonction devra valider le test suivant :

```
void testToStringParc(){
  Parc p = newParcVide();
  Vehicule v1 = newVehicule("Yaris", 2020);
  Vehicule v2 = newVehicule("Touran", 2019);
  ajouterVehicule(p, v1);
  ajouterVehicule(p, v2);
  assertEquals("Yaris␣2020\nTouran␣2019\n", toString(p));
}
```

Q8 : Nombre de voitures d'un certain modèle

Écrire la fonction `int nbVoitures (Parc p, String modele)` qui retourne le nombre de véhicules d'un certain modèle dans le parc de voitures.

Q9 : Fusion de deux parcs de véhicules

Écrire la fonction `boolean fusionParcs (Parc p1, Parc p2)` qui déplace un maximum de véhicules du parc p2 au parc p1 dans la limite des places disponibles de p1. C'est-à-dire que si p2 contient plus de véhicules que ne peut en contenir p1, on déplace le maximum de véhicule dans p1 et on s'arrange pour que p2 reste bien organisé (ie. sans trou).

Q9 : Suppression d'un véhicule

Écrire la fonction `void suppressionVehicule (Parc p, Vehicule v)` qui supprime la première occurrence du véhicule v dans le parc p. Attention, les véhicules doivent toujours être sur le début du tableau (pas de trou dans le tableau).