Classe Iterable

Resp. UE : Érik Martin-Dorel & Jean-Paul Bodeveix

Sujets POO: Christelle Chaudet & Jean-Paul Bodeveix

1 Préparation de l'environnement de développement

Dans les TPs nous utiliserons obligatoirement :

- l'IDE Eclipse avec le plugin SonarLint,
- l'historisation du développement avec Git.

Assurez-vous d'avoir les outils nécessaires (IDE Eclipse, terminal pour l'utilisation de GIT, un compte GitHub), sinon vous trouverez les documents d'installation sur Moodle.

2 Sujet

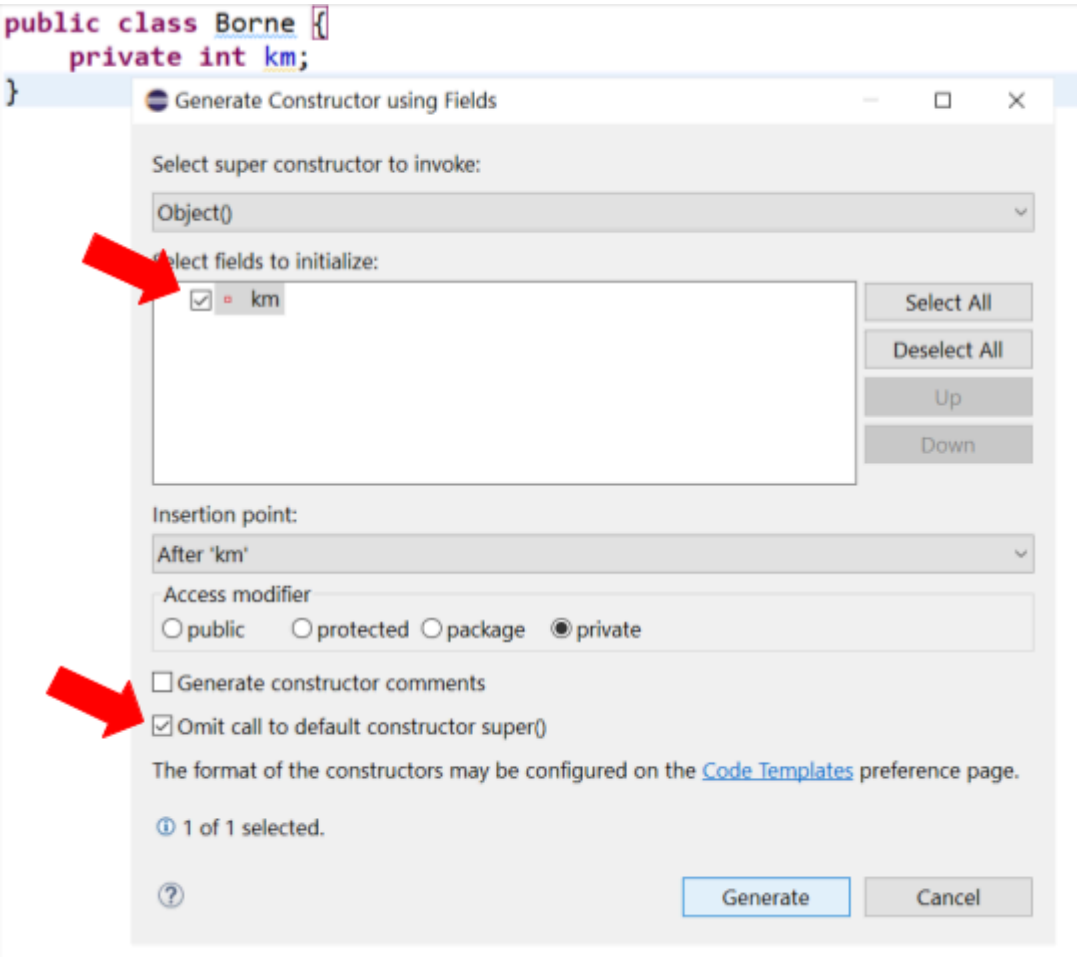
Cette série de TP a pour but de réaliser un jeu de milles bornes où les joueurs sont gérés par la machine selon diverses stratégies. Les règles du jeu sont décrites ici : règles du jeu. Les TPs introduisent progressivement les éléments constitutifs du jeu en exploitant les notions vues en cours.

Les Cartes

En utilisant les outils de génération automatique d'Eclipse, définir les classes du diagramme en page 4 avec leurs constructeurs et accesseurs en lecture.

Rappel :

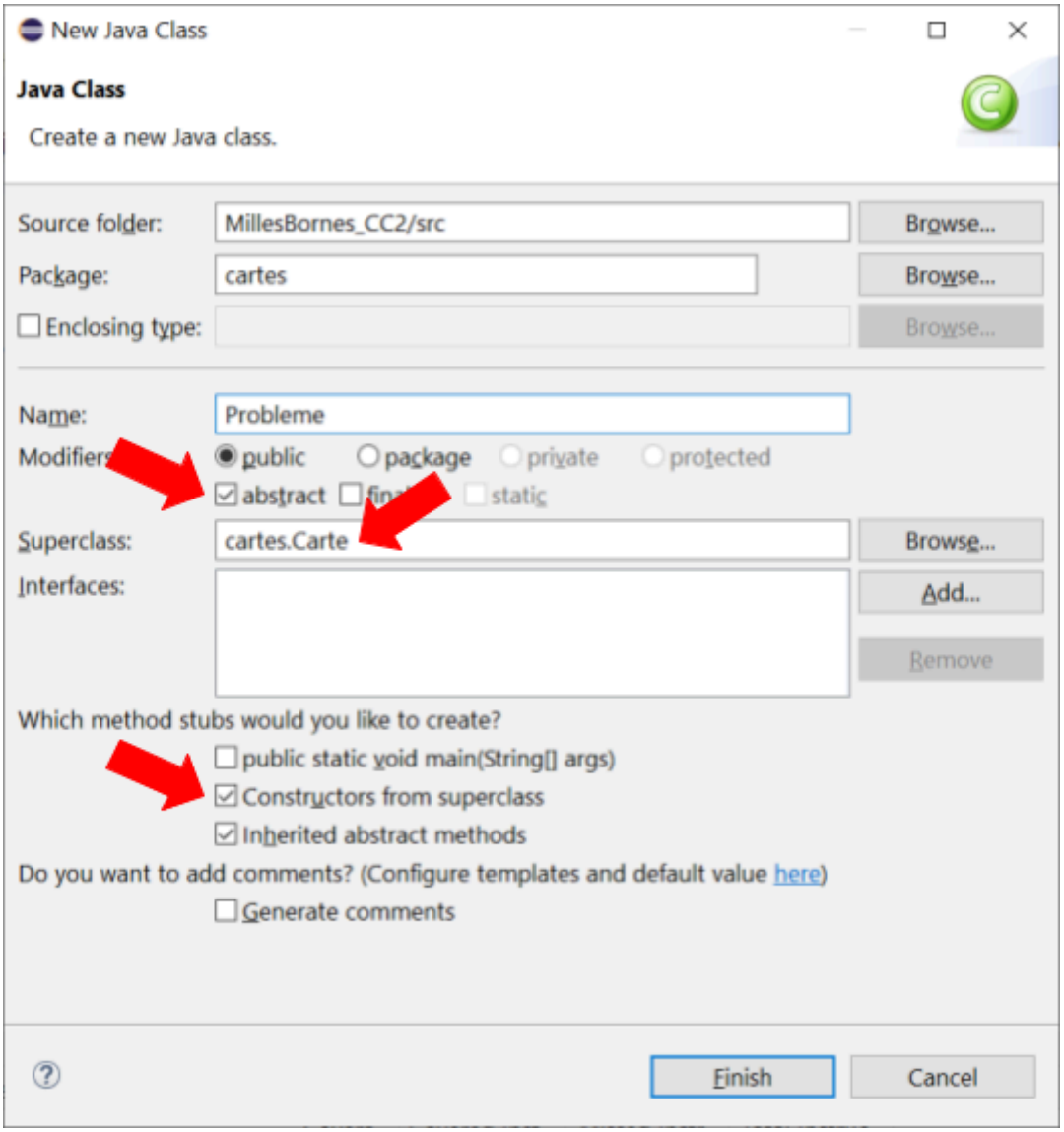
Utiliser la génération automatique des constructeurs et des des getters.



Pour les getters soit il y en a plusieurs et vous passez par Source > Generate Getters and setters... et cochez ceuw que vous souhaitez générer (les getters ou les setters ou les deux).

Soit il y en a peu et vous tapez directement dans votre code : get + Ctrl + espace

Lors de la création de la classe, sélectionnez les cases qui vous intéressent.



Sélectionner ou non abstract.

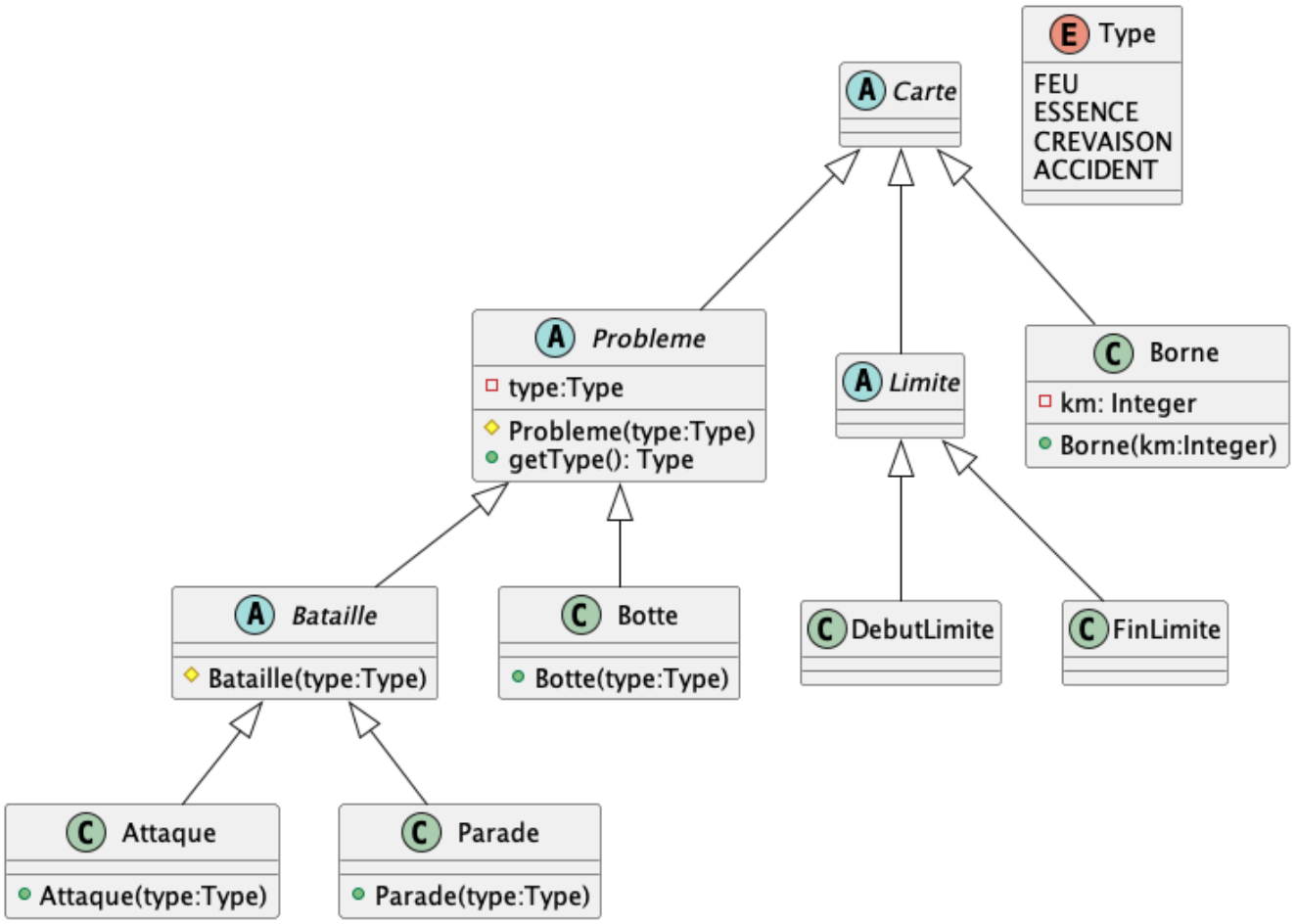
La superclasse est par défaut la classe **Object**. Tapez le début du nom de la classe mère (ex : Ca + Crt + espace), Eclipse trouvera le nom du paquetage et le nom de la classe.

Si la classe mère a un constructeur, cochez “Constructors from superclass”.

Si la classe mère possède des méthodes abstraites, cochez “Inherited abstract methods”

Travail à effectuer :

1. Implémenter dans le package **cartes** les classes du diagramme ci-dessous.

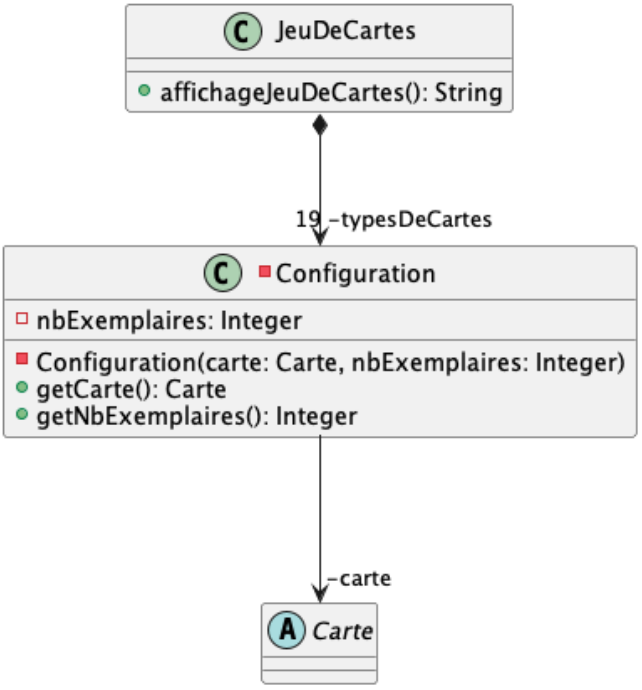


2. Définir les méthodes **toString** renvoyant le nom de chaque carte en s'inspirant de la description du contenu d'un jeu donnée dans règles du jeu. On nommera **Feu rouge** et **Feu vert** les attaques et parades représentées par des feux. On considèrera que **Prioritaire** est une botte associée au feu et non à une limite de vitesse. Pour faciliter la redéfinition de **toString** dans les sous-classes de **Probleme** on ajoutera 3 paramètres au constructeur de l'énumération Type, contenant respectivement l'affichage souhaité lorsque la carte de ce type est une attaque, une parade ou une botte.

   On s'inspirera de l'exemple des planètes de la JavaDoc : tutorial enum

Le Jeu de Cartes

1. Ajouter au package **cartes** la classe **JeuDeCartes** contenant un tableau de configurations. Chaque configuration contient une carte et son nombre d'exemplaires.



La méthode **affichageJeuDeCartes** permet d'afficher pour chacune des cartes son nombre d'exemplaires dans le jeu :

JEU :

10 25KM
10 50KM
10 75KM
12 100KM
4 200KM
14 Feu Vert
6 Fin Limite
6 Bidon d'essence
6 Roue de secours
6 Réparation
5 Feu Rouge
4 Limite 50
3 Panne d'essence
3 Crevaison
3 Accident
1 Prioritaire
1 Citerne
1 Increvable
1 As du volant

2. Définir dans la classe JeuDeCartes public Carte[] donnerCartes() renvoyant un tableau contenant toutes les cartes répliquées selon le nombre d'exemplaires indiqué dans la configuration.

3. Ajouter dans un package **testsFonctionnels** la classe **TestJeuDeCartes**. Écrire la méthode réalisant l'affichage précédent (sans prendre en compte les colonnes !).

Le Sabot

Cette partie est réalisable durant la séance de TP. Si vous ne l'avez pas terminée à la fin des deux heures vous devrez la terminer chez vous avant la séance suivante.

1. Ajouter un package jeu contenant la classe Sabot contenant des cartes stockées dans un tableau fourni par le constructeur. L'attribut **nbCartes** indiquera le nombre effectif de cartes du sabot, initialement toutes les cartes du jeu.

    a. Définir la méthode **estVide** indiquant si la pioche est vide.
    b. Définir la méthode **ajouterCarte** ajoutant une carte. Une exception sera levée en cas de dépassement de capacité.
    c. Rendre la classe itérable : on gèrera les exceptions **IllegalState** et **ConcurrentModification**. La méthode remove devra être définie.
    d. Utiliser un itérateur pour définir la méthode **piocher** renvoyant et supprimant la première carte du sabot.

2. Ajouter dans le package **testsFonctionnels** la classe **TestSabot** contenant un programme de test créant un sabot et prélevant et affichant ses cartes jusqu'à ce qu'il soit vide.
    a. On utilisera **piocher**. L'affichage attendu est le suivant :

```
je pioche 25KM
je pioche 25KM
…
```

toutes les cartes du jeu doivent apparaître.

    b. On utilisera un itérateur et remove pour obtenir le même résultat
    c. On ajoute à la boucle b) un appel à piocher. Une exception doit être levée. De même, après avoir pioché une carte avant la boucle (afin d'éviter un débordement du tableau **cartes**), insérer l'**asDuVolant** dans la boucle doit lever une exception.