Fiche méthodologique de projet

C.7

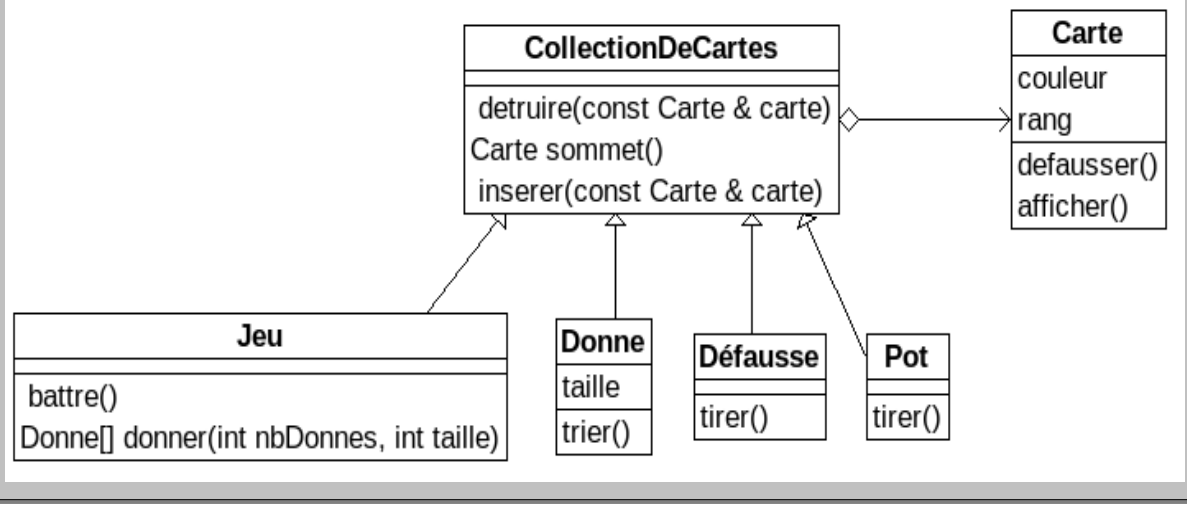
Établir un diagramme de classes de conception

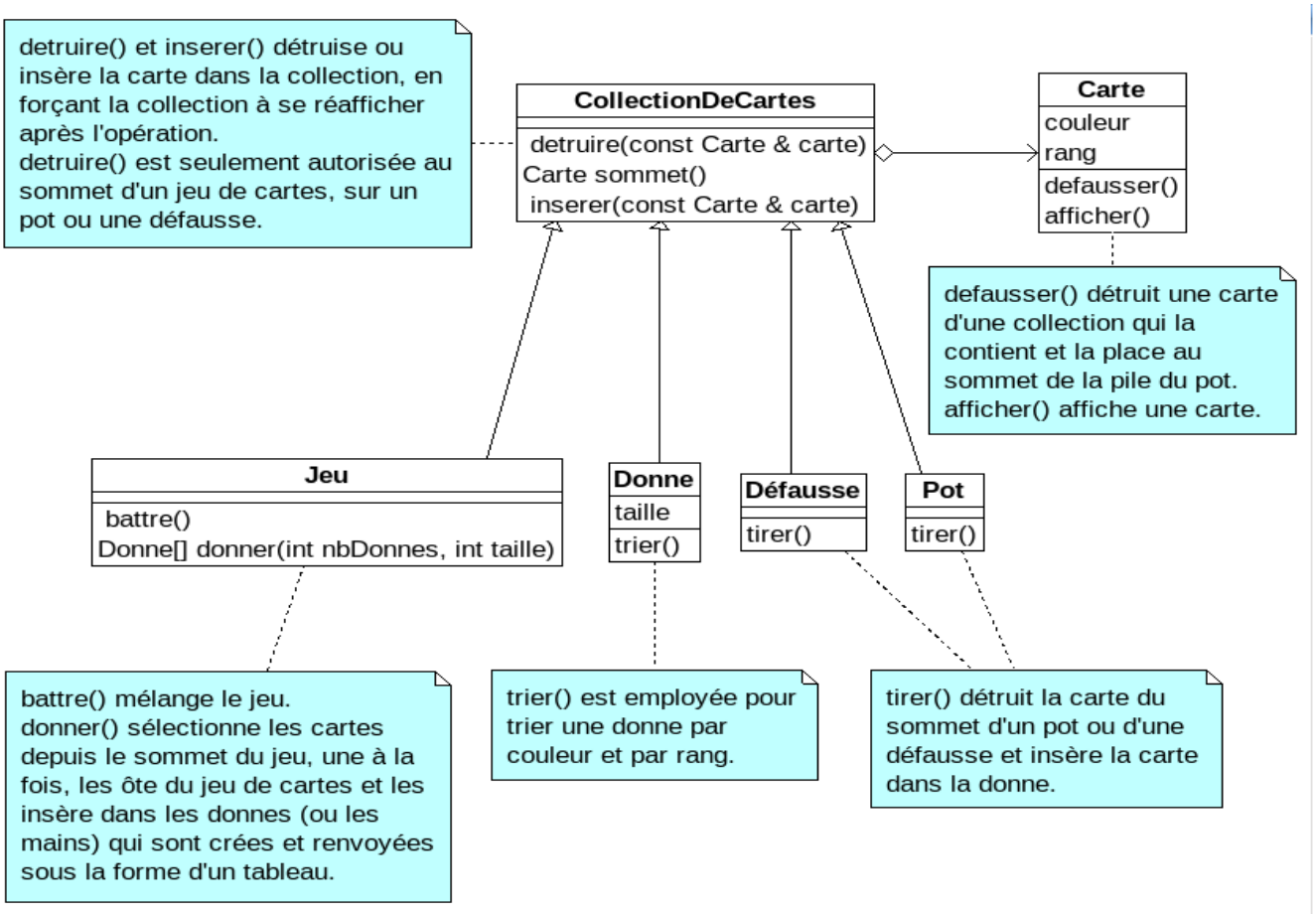
Objectif

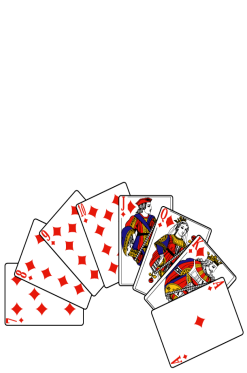
Rédiger les documents nécessaires à la compréhension d'un diagramme de classes de conception : démarche, choix, rôles, etc ...

Exemple

Problème : ne pas seulement fournir un diagramme de classes « brut » !







Conception

En établissant une première version du diagramme, on a trouvé des « ressemblances » pour les classes Jeu, Defausse et Pot. En effet, ces trois classes modélisent une collection de cartes sous la forme d'une **pile** (de cartes) avec un sommet.

On introduit donc une nouvelle classe (purement informatique) : **Pile**. Elle sera probablement abstraite et mettra en oeuvre le polymorphisme.

Pattern Polymorphisme

Il faut utiliser le polymorphisme pour **implémenter les variations de comportement en fonction de la classe**. Le polymorphisme est le concept idéal pour réaliser une variation du comportement des objets en fonction de leur type. Il est utile dans le cadre de prévisions de remplacement ou d'ajout de composants. Les programmes utilisant des (nombreuses) instructions if/elseif ou switch/case nécessitent une intervention au niveau du client lorsqu'on souhaite ajouter un cas. Cela rend le programme plus difficile à maintenir et le module difficile à réutiliser.

Exemple

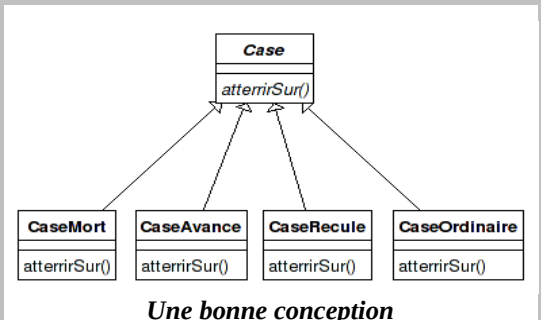
Problème : comment concevoir les actions en fonction des types de cases ?

Pour résumer, lorsqu'un joueur atterrit sur une case celle-ci influe sur le comportement du déplacement du pion : il existe une case qui fait revenir au départ, une autre qui fait faire un saut de trois cases, etc ... Il existe donc une action selon le type de cases.

Une mauvaise conception (trop souvent réalisée) aboutirait à ce genre de code :

```
#define CASE_MORT 1
//etc ...

switch(case.type)
{
case CASE_MORT :  revenirAuDepart(); break;
case CASE_AVANCE :   avancer(3); break;
case CASE_RECULE :   reculer(3); break;
// etc ...
case CASE_NORMALE :  break; //ne rien faire !
}
```



Une bonne conception

En appliquant Polymorphisme, on va créer une classe pour chaque sorte de Case dont la responsabilité atterrirSur diffère. On implémentera donc une méthode atterrirSur() dans chacune d'elles.