Projet de développement Java
Version de base du jeu *Citadelles*
partie 1

P. Albers

17 novembre 2023

Table des matières

1 Introduction

Le projet de développement a pour but d'implémenter une application en utilisant la plupart des aspects de la programmation objet Java, ainsi que tous les éléments de la gestion de projet.

L'application que vous devrez implémenter est la version électronique du jeu de plateau *Citadelles* de Bruno Faidutti (édition *Edge Entertainment*). L'objectif du projet est uniquement à but éducatif, et ne pourra en aucun cas déboucher sur une utilisation commerciale à but lucratif ou non.

Ce jeu a été choisi dans la mesure où il ne nécessite pas une interface graphique poussée qui prendrait trop de temps à implémenter. L'accent est mis dans ce projet sur la mise en pratique des différents aspects de la programmation orientée objet en Java.

Le développement comportera plusieurs versions :

1. la version de base qui sera une version texte du jeu (c'est à dire sans graphisme) et en utilisant une configuration de base décrite dans la présentation du jeu ; il y aura quatre joueurs dont trois seront simulés par l'application (les choix seront pris de manière aléatoire, *i.e.* sans intelligence),
2. l'extension *version complète* qui devra implémenter l'ensemble des personnages, l'ensemble des cartes *merveille*, ainsi que l'utilisation du jeu de 2 à 9 joueurs comme dans le jeu de plateau,
3. l'extension dite *intelligente* qui devra utiliser des algorithmes issus de la théorie des jeux pour la simulation des joueurs,
4. l'extension *multi-joueurs* qui devra permettre en place la possibilité d'utiliser l'application en réseau avec plusieurs joueurs,
5. l'extension *graphique* qui utilisera les outils graphiques de base de Java pour une version plus conviviale.

2 Les classes du modèle

Dans la version de base, il vous est demandé de mettre en place les classes nécessaires pour représenter les données du jeu. Cinq classes principales vont être définies : *Quartier*, *Joueur*, *Personnage*, *Pioche* et *PlateauDeJeu*. La figure 1 présente la spécification de ces cinq classes et leurs relations.

2.1 Les cartes *Quartier*

La classe *Quartier* représente un quartier à bâtir. Un quartier possède quatre attributs : un nom, un coût de construction, des caractéristiques et un type, conformément au diagramme de classe de la figure 1. Les caractéristiques correspondent au texte que l'on peut trouver sur les cartes de jeu ; elles sont destinées à l'affichage lorsque l'utilisateur le demandera pour information. On définit par ailleurs une constante $TYPE\_QUARTIERS$ qui contient les valeurs que peut prendre le champ *type*.

Un seul constructeur prend les quatre valeurs en paramètres. À ce contructeur, il faut ajouter les accesseurs à ces attributs en lecture et en écriture (4 attributs ⇒ 8 accesseurs).

Travail à faire :

1. Créez un nouveau projet *citadelles*
2. Ajoutez un nouveau paquetage *modele*.
3. Ajoutez dans ce paquetage la classe *Quartier*.
4. Ajoutez les attributs privés *nom*, *type*, *coutConstruction* et *caracteristiques*. La constante $TYPE\_QUARTIERS$ doit être déclarée en *public static final*.
5. Implémentez les accesseurs en lecture et écriture de l'attribut *nom* comme suit :

```
public String getNom(){
     return this.nom;
}
public void setNom(String nom){
     this.nom = nom;
}
```

6. Implémentez les accesseurs de l'attribut *type* : $getType()$ et $setType(String)$. L'accesseur en écriture $setType(String)$ doit vérifier que l'argument correspond bien à l'une des valeurs de $TYPE\_QUARTIERS$; si ce n'est pas le cas, *type* prend comme la valeur la chaîne vide "".
7. Implémentez les accesseurs de l'attribut *coutConstruction* : $getCout()$ et $setCout(int)$. Ce dernier vérifie si le coût est bien compris entre 1 et 6 inclus ; si ce n'est pas, *cout* prendra la valeur 0.
8. Implémentez les accesseurs de l'attribut *caracteristiques* : $getCaracteristiques()$ et $setCaracteristiques(String)$.
9. Implémentez le constructeur tel que défini dans la figure 1. Vous préférerez appeler les accesseurs en écriture de chaque attribut pour les initialiser.

Test de la classe *Quartier*

Il est préférable de tester les classes au fur et à mesure du développement. Dans la mesure où vous travaillez en équipe, ces tests doivent être consignés dans un seul et même fichier afin de centraliser l'information.

Vous utiliserez pour cela la classe *Test.java* qui devra se trouver dans le paquetage *test*. La méthode de classe *test*(*boolean*, *String*) sera appelée par chaque classe de test : elle affichera si le test est correct ou non, ainsi que la ligne du fichier où elle a été appelée. Le premier paramètre sera le résultat du test à vérifier ; le second un message destiné à l'utilisateur pour connaître le but du test.

Travail à faire :

1. Téléchargez le fichier *cahierDeTests.xls* depuis *Moodle*.
2. Ajoutez un nouveau paquetage *test* à votre projet.
3. Téléchargez depuis *Moodle* le fichier *TestQuartier.java* et *Test.java*, et ajoutez le dans ce paquetage.
4. Testez à partir de *TestQuartier.java* chacune des méthodes de la classe *Quartier* en appelant les différentes méthodes de test.
5. Indiquez dans le cahier de tests le résultat de chaque test, qu'il fonctionne ou non. Lorsque le test ne fonctionne pas, il est nécessaire de modifier le code et de refaire le test tant que le résultat n'est pas celui attendu.

2.2 Les joueurs

La classe *Joueur* possède les éléments suivants : le nom du joueur, son trésor (*i.e.* le nombre de pièces d'or qu'il possède), sa main constituée d'un nombre de cartes quartier et l'ensemble de cartes quartier constituant sa cité. L'attribut *possedeCouronne* indique si le joueur commencera au tour suivant.

Il est choisi pour représenter la main d'utiliser une instance de la classe *ArrayList<>* qui permet d'implémenter des tableaux redimentionnables. Cette classe est générique et permet de manipuler tout type d'objet.

Le nombre de quartiers constituant la cité est au maximum de huit, mais il peut y avoir moins. Il est choisi d'utiliser un tableau pour représenter cet ensemble de quartiers. L'entier *nbQuartiers* permet de connaître le nombre de quartiers dans la cité à tout moment.

Implémentation de la classe

Travail à faire :

1. Ajoutez au paquetage *modele* la classe *Joueur.java*.
2. Déclarez les six attributs définis dans la figure 1. L'attribut *cite* est un tableau de *Quartier*, et *main* est une variable de type *ArrayList<Quartier>*.
3. Implémentez le constructeur de la classe qui prend uniquement en paramètre le nom du joueur. Le trésor et le nombre de quartiers de la main seront initialisés à 0. L'attribut *possedeCouronne* sera initialisé à *false*. Le tableau *cite* aura une taille maximale de 8. La main sera initialisée comme suit : *this.main = new ArrayList<Quartier>()*
4. Ajoutez les accesseurs en lecture :
   — *getNom()* qui renvoie le nom du joueur,
   — *nbPieces()* qui renvoie le nombre de pièces d'or du joueur,
   — *nbQuartiersDansCite()* qui renvoie le nombre quartiers de la cité,
   — *getCite()* qui renvoie le tableau complet de quartier de la cité,
   — *getMain()* qui renvoie l'ensemble de la main *ArrayList<Quartier>*
   — *nbQuartiersDansMain()* qui renvoie le nombre de quartiers de la main [a]
   — et *getPossedeCouronne()* qui renvoie la valeur de l'attribut *possedeCouronne*.
   ainsi que l'accesseur en écriture :
   — *setPossedeCouronne(boolean b)* qui modifie la valeur de l'attribut *possedeCouronne*.
5. Implémentez les méthodes *ajouterPieces(int)* et *retirerPieces(int)* qui permettent repectivement d'ajouter et retirer des pièces d'or du trésor du joueur. Vérifiez que le nombre de pièces passés en paramètre est positif. D'autre part, il faut s'assurer que l'on ne peut pas retirer plus de pièces que ne comporte le trésor.
6. Ajoutez la méthode *ajouterQuartierDansCite(Quartier)* en vérifiant que le tableau n'est pas déjà plein. Le fait que l'on ne peut pas ajouter deux quartiers identiques dans une cité ne sera pas traité dans cette méthode.
7. Ajoutez la méthode *quartierPresentDansCite(String)* qui renvoie *true* si le quartier dont le nom est passé en paramètre, est présent dans la cité.
8. Implémentez *retirerQuartierDansCite(String)* qui retire le quartier de la cité dont le nom est donné en paramètre. Cette méthode renvoie le quartier retiré, ou *null* si le quartier n'est pas présent.
9. Ajoutez *ajouterQuartierDansMain(Quartier)* qui permet d'ajouter un quartier à la main du joueur.
10. Ajoutez *retirerQuartierDansMain()* dont le but est de retirer un quartier de la main de manière aléatoire. Vous pourrez utiliser pour cela la classe *Random* comme suit :

    ```
    Random generateur = new Random();
    int numeroHasard = generateur.nextInt(this.nbQuartiersDansMain());
    ```

    Si la main est vide, la méthode renverra *null*.
11. Implémentez *reinitialiser()* qui remettra à 0 le trésor du joueur, videra sa main et sa cité.

a. Pour connaître les méthodes associées à la classe *ArrayList*, veuillez vous reporter à la *JavaDoc* : https ://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/ArrayList.html

Test de la classe

Travail à faire :

1. Téléchargez le fichier *TestJoueur* depuis *Moodle*, puis ajoutez le dans le paquetage *test* de votre projet. Exécutez les méthodes une à une et répertoriez les résultats dans votre cahier de tests.

2.3 La classe *Caracteristiques*

La classe *Caracteristiques* vous est donnée, et comporte les caractéristiques indiquées sur chaque carte Personnage du jeu. Ses données pourront être affichées durant le jeu, si l'utilisateur le demande.

À chaque personnage, correspond une chaîne de caractères qui est une variable de classe (mot clef *static*). Ce qui veut dire que ces variables sont globales à la classe et ne pourront pas être associées à des instances.

Elles s'utilisent en les appliquant directement sur la classe par exemple comme suit :

```
String s = Caracteristiques.ASSASSIN;
```

Travail à faire :

1. Téléchargez le fichier *Caracteristiques.java* depuis la plate-forme *Moodle*.
2. Ajoutez ce fichier au paquetage *modele*.

2.4 La classe *Personnage*

Elle possède les caractéristiques suivantes : le nom du personnage, son rang, la description de son personnage (caractéristiques du personnage), le joueur auquel le personnage appartient, le fait de savoir si le personnage est assassiné ou volé. La description du personnage pourra s'afficher à l'écran pour informer l'utilisateur de l'application.

La classe *PlateauDeJeu* contiendra un ensemble de personnages. Afin de pouvoir accéder aux informations du plateau de jeu depuis la classe *Pesonnage*, on ajoute pour cela une variable *plateau*. Ce lien se fera lors de l'ajout des personnages choisis dans le plateau de jeu. Les méthodes *setPlateau* et *getPlateau* seront ajoutés lorsque la classe *PlateauDeJeu* sera implémentée.

Outre les différents accesseurs de la classe, un personnage pourra ajouter des pièces au personnage auquel il appartient, lui ajouter un quartier dans la main du joueur, construire un nouveau quartier dans sa cité, percevoir des ressources spécifiques et utiliser ses pouvoirs.

Pour implémenter un personnage particulier, il faudra donc écrire une sous classe de *Personnage*. Comme les pouvoirs sont particuliers à chaque personnage, il est impossible d'implémenter une méthode de manière générique. C'est pourquoi la méthode *utiliserPouvoir*() sera déclarée abstraite.

Travail à faire :

1. Ajoutez au paquetage *modele* la classe *Personnage.java*.
2. Déclarez les attributs *nom*, *rang*, *caracteristiques*, *joueur assassine* et *vole*, conformément à la figure 1. L'attribut *plateau* sera ajouté plus tard (partie 2.7).
3. Implémentez le constructeur de la classe qui prend uniquement en paramètre le nom du personnage, son rang (de type entier) et ses caractéristiques (une chaîne de caractères). L'attribut *joueur* est initialisé à la valeur *null* pour indiquer que ce personnage n'est attribué à aucun joueur. Les attributs *vole* et *assassin* prennent la valeur $false$ pour indiquer que le personnage n'est ni volé, ni assassiné.
4. Ajoutez les accesseurs en lecture :
   — $getNom()$ qui renvoie le nom du personnage,
   — $getRang()$ qui renvoie son rang,
   — $getCaracteristiques()$ qui renvoie ses caractéristiques,
   — $getJoueur()$ qui renvoie le joueur auquel le personnage est attribué,
   — $getAssassine()$ qui renvoie si le personnage est assassiné ou non,
   — $getVole()$ qui renvoie si le personnage est volé ou non,
5. Ajoutez les trois accesseurs en écriture [a] :
   — $setJoueur(Joueur\ j)$ qui associe le joueur j au personnage,
   — $setVole()$ qui met l'attribut *vole* à *true*,
   — et $setAssasine()$ qui met l'attribut *assassine* à *true*.
6. Implémentez la méthode $ajouterPieces()$ qui permet d'ajouter deux pièces au trésor du joueur auquel le personnage est associé. Il vous faudra vérifier le cas où le personnage n'est associé à aucun joueur (*i.e.* l'attribut *joueur* vaut *null*), et que le personnage n'a pas été assassiné ; dans ces deux cas, on ne peut ajouter de pièces au trésor.
7. Implémentez la méthode $ajouterQuatier(Quartier\ nouveau)$ qui permet d'ajouter un nouveau quartier à la main du joueur auquel le personnage est associé. Gérez également dans cette méthode les deux cas particuliers sités dans la question précédente.
8. Implémentez la méthode $construire(Quartier\ nouveau)$ qui ajoute un nouveau quartier dans la cité du joueur. Gérez les deux cas particuliers cités auparavant.
9. Implémentez la méthode $percevoirRessourcesSpecifiques()$ qui affichera par défaut le message *aucune ressource spécifique*. En effet, certain personnage ne reçoivent aucune ressource spécifique. Veuillez également gérer les deux cas particuliers sités auparavant.
10. Déclarer la méthode abstraite $utiliserPouvoir()$.
11. Ajoutez enfin la méthode $reinitialiser()$ qui permet de remettre à leur valeur initiale (comme dans le constructeur) les attributs *joueur*, *vole* et *assassine*. Cette méthode sera notamment appelée entre chaque tour avant la réattribution des personnages aux joueurs.

a. Les autres attributs ne seront pas modifiés une fois le personnage créé.

2.5 La classe *Roi* :

La classe *Roi* est une sous classe de *Personnage*. Un roi :
— perçoit pour chaque quartier noble qu'il possède dans sa cité une pièce d'or,

— a comme pouvoir de prendre la couronne. Le joueur à qui ce personnage est associé pourra choisir son personnage en premier au prochain tour.

Travail à faire :

1. Ajoutez au paquetage *modele* la classe *Roi.java* qui hérite de la classe *Personnage* conformément au diagramme de la figure 2.
2. Implémentez le constructeur vide (*i.e.* sans paramètres) qui appelle le constructeur de la sur classe en spécifiant en paramètres le nom du personnage ("Roi"), son rang (4) et ses caractéristiques (valeur de la variable $Caracteristiques.ROI$).
3. Implémentez la méthode $utiliserPouvoir()$ qui affichera à l'écran le message $"Je\ prends\ la\ couronne"$, et qui mettra à vrai l'attribut *possedeCouronne* du joueur.
4. Surchargez (*i.e.* réécrire) la méthode $percevoirRessourcesSpecifiques()$. Dans cette méthode, il faut comptabiliser le nombre de quartiers nobles de la cité du joueur associé au personnage du *Roi* et ajouter ce même nombre au trésor du joueur. Vous afficherez également un message pour indiquer le nombre de pièces ajoutées.
5. Téléchargez depuis *Moodle* le fichier *TestRoi.java* et ajoutez le dans le paquetage *test*. Ce test permet de vérifier toutes les fonctionnalités à la fois de la classe *Personnage* et de la classe *Roi*.

   Exécutez les méthodes une à une et répertoriez les résultats dans votre cahier de tests.

2.6 La pioche

Lors du jeu, il est nécessaire d'avoir une pioche constituée de cartes Quartier. Les joueurs pourront prendre une ou plusieurs en haut de la pioche. Certaines cartes seront à l'inverse remises dans le fond de la pioche.

Par conséquent, la pioche revient à faire une file de quartiers (comme une file d'attente devant un guicher). En algorithmique, on parle de liste $FIFO$ (*first in first out* - premier arrivé, premier sorti).

Pour sauvegarder les éléments de la pioche, nous utiliserons une variable *liste* de type $ArrayList < Quartier >$. On prendra comme sommet de la pioche le premier élément de *liste*.

Travail à faire :

1. Ajoutez au paquetage *modele* la classe *Pioche.java* présentée dans le diagramme de la figure 3.
2. Implémentez le constructeur vide qui initialise la variable *liste*.
3. Implémentez les méthodes :
   — *piocher*() qui retire la carte au sommet de la pioche et la renvoie ; si la pioche est vide, la méthode renverra *null*,
   — *ajouter*(*Quartier nouveau*) qui ajoute une carte Quartier au bas de la pioche,
   — *nombreElements*() qui renvoie le nombre d'éléments de la pioche,
   — *melanger*() qui permet de mélanger la pioche.
   Pour implémenter la méthode *melanger*, une idée est de générer de manière aléatoire deux indices entre 0 et le nombre d'éléments de la pioche et d'intervertir les deux quartiers. Vous pourrez utiliser pour cela les méthodes *set*() et *get*() de *ArrayList*, ainsi que la classe *Random* comme suit :

```
Random generateur = new Random();
...
int i = generateur.nextInt(nbEltsPioche);
int j = generateur.nextInt(nbEltsPioche);
...
```

4. Téléchargez depuis *Moodle* le fichier *TestPioche.java* et ajoutez le dans le paquetage *test*. Exécutez les méthodes une à une et répertoriez les résultats dans votre cahier de tests.

2.7 La classe *Plateau*

Le plateau de jeu comporte une liste de personnages qui ont été choisis au début de la partie, d'une liste de joueurs, et de la pioche de cartes *Quartier*.

Travail à faire :

1. Ajoutez au paquetage *modele* la classe *PlateauDeJeu.java*.
2. Déclarez les cinq attributs définis dans la figure 1.
3. Implémentez le constructeur de la classe qui ne prend aucun paramètre.
   Comme les parties peuvent contenir 9 joueurs au maximum, le tableau *listeJoueurs* aura une tête de neuf maximale. De même, comme les parties contiennent 8 ou 9 personnages, le tableau *listePersonnages* sera initialisé avec une taille de 9. À la création du plateau de jeu, il n'y aura aucun personnage ni joueur.
   En ce qui concerne la pioche, vous devrez créer une instance de la pioche dans ce constructeur (lien de composition dans le diagramme de classe).
4. Ajoutez dans la classe *Personnage*, l'attribut *plateau* ainsi que les accesseurs *getPlateau*() et *setPlateau*().
5. Ajoutez les accesseurs en lecture :
   — *getNombrePersonnages*() qui renvoie la valeur *nombrePersonnage*,
   — *getNombreJoueurs*() qui renvoie la valeur *nombreJoueurs*,
   — *getPioche*() qui renvoie la variable *pioche*,
   — *getPersonnage*(*int i*) qui renvoie le i^{ème} personnage du tableau *listePersonnages*,
   — *getJoueur*(*int i*) qui renvoie le i^{ème} personnage du tableau *listeJoueurs*.
   Dans les deux dernières méthodes, les indices du tableau seront compris entre 0 et le nombre d'éléments du tableau moins 1. Dans le cas où la variable i n'est pas dans cet interval, ces deux méthodes drvroint renvoyer la valeur *null*.
6. Implémentez les méthodes *ajouterPersonnage* et *ajouterJoueur* qui permettent d'ajouter respectivement un nouveau personannage et un nouveau joueur dans le plateau de jeu. Vous vérifierez que les valeurs passés en paramètre ne sont pas *null* et que les tableaux ne sont pas déjà pleins.
   La méthode *ajouterPersonnage* fera en supplément l'association du plateau au personnage ajouté via la méthode *setPlateau* de la classe *Personnage*.
7. Téléchargez depuis *Moodle* le fichier *TestPlateauDeJeu.java* et ajoutez le dans le paquetage *test*. Exécutez les méthodes une à une et répertoriez les résultats dans votre cahier de tests.

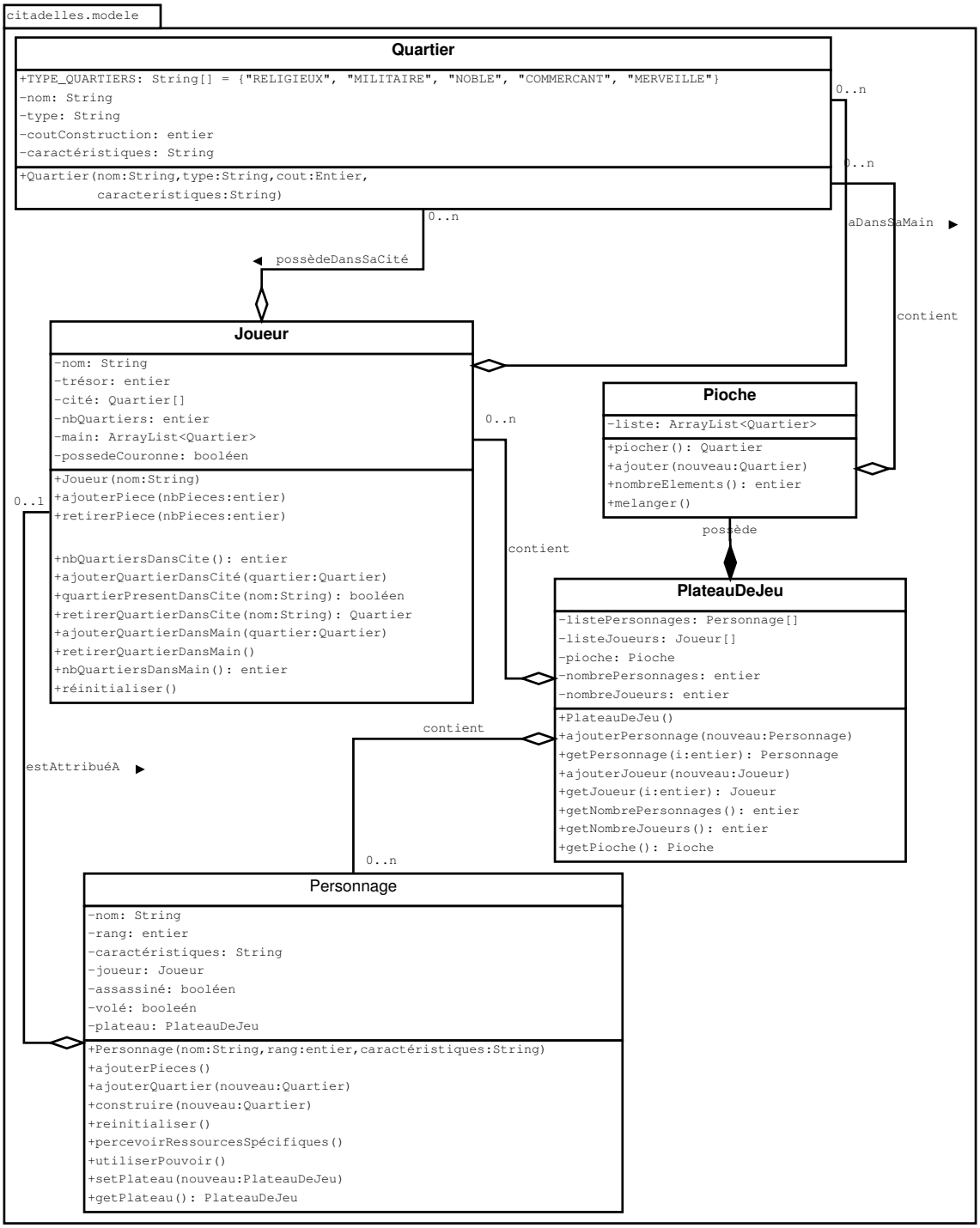

FIGURE 1 – Diagramme principal de classes

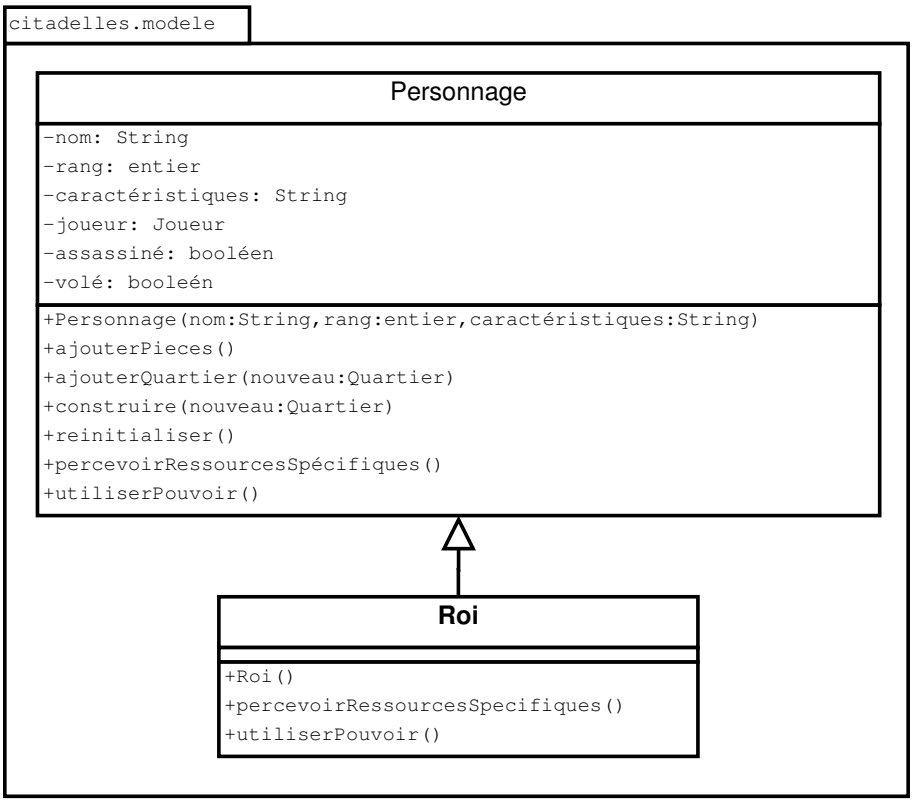

FIGURE 2 – Diagramme de la classe *Roi*

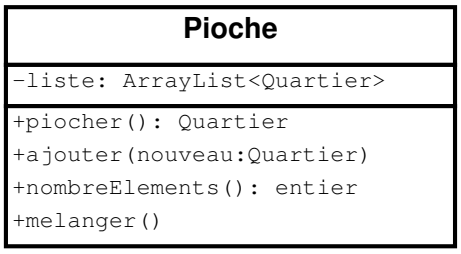

FIGURE 3 – Diagramme de la classe *Pioche*