PRIOU
Roger

RAPPORT

7.2.1 – Scanner :

L'objet Scanner permet de récupérer le texte tapé par l'utilisateur dans le terminal pour pouvoir le traiter. Pour récupérer correctement le texte il faut que le « clavier » soit passé en paramètre pour créer un objet Scanner. Ensuite dans une variable on stocke la ligne tapée dont on extrait les commandes pas la suite.

7.5 – printLocationInfo :

Cette méthode permet d'afficher toutes les informations sur la Room courante, ainsi au lieu d'utiliser plusieurs méthodes pour les différentes informations, on regroupe tout dans celle-ci évitant ainsi la duplication de code qui nuit à un bon développement de programme.

7.6 – getExits :

Dans cet exercice nous avons mis à jour notre jeu pour qu'il possède un couplage moins fort qu'auparavant. Le niveau de couplage entre les classes est proportionnel au lien qu'elles ont entre elles, autrement dit la capacité de chacune à appeler l'autre. Pour cela nous allons utiliser l'encapsulation qui ne révèle qu'aux autres classes les informations nécessaire au bon fonctionnement, c'est-à-dire seulement ce que peut faire la classe. Ainsi nous devons créer un accesseur getExits qui va nous permettre de récupérer chaque sortie d'un objet de type Room en temps voulu.

7.7 – getExitString :

Cette méthode n'est rien d'autre qu'une fonction permettant de retourner l'ensemble des sorties disponibles sous la forme d'une chaîne de caractère. Donc pour ce faire elle se base sur l'accesseur getExits lui-même.

7.8 – HashMap aExits:

Le concept HashMap permet de faciliter grandement la création de Room et des sorties qui vont avec, ainsi que le tout le travail autour de ces objets. Une HashMap est un tableau qui à chaque « clé » ou « key » associe une valeur, c'est pour cela qu'elle est très utile car elle va permettre pour chaque Room d'associer à chacune de ses sorties un autre objet Room si nécessaire. Donc pour résumé dans notre cas les « clés » du HashMap correspondent à des directions et les valeurs à des objets de type Room.

7.8.1 – Déplacement vertical :

Ayant précédemment créé une Hashmap, nous pouvons aisément insérer des déplacements verticaux autres que Nord, Sud, Est, Ouest. En effet il nous suffit d'entrer comme clé « Bas » ou « Haut » ou ce que l'on veut comme déplacement et d'y ajouter la valeur de sortie, ici une Room.

7.9 – Keyset :

Cette méthode fait partie intégrante de la classe Hashmap et elle a pour fonction de stocker toutes les clés de l'objet Hashmap correspondant ainsi que la valeur qui lui est associé.

7.10 – getExitString v2 :

Après avoir étudié le fonctionnement de la méthode keyset() nous sommes en mesure de l'utiliser pour améliorer le corps de la méthode permettant d'afficher les sorties disponibles pour chaque Room.

7.11 – getLongDescription :

Cette méthode correspond à un accesseur permettant de réduire encore plus le couplage de la classe. Ainsi nous avons créé un accesseur permettant d'afficher l'ensemble des informations de l'objet de type Room en question, tel que le nom (aDescrption) ou encore les sorties disponibles (getExitString). En effet il nous suffira d'appeler cet accesseur dans la classe Game à chaque tour de boucle du jeu pour afficher ces informations et non de manipuler les objets Room en eux-même.

7.12 – SCHÉMA :

Cf. ANNEXES

7.14 – look :

Nous voulions rajouter une commande permettant d'afficher à souhait les informations d'une Room. Pour cela nous avons dû ajouter la nouvelle commande dans les commandes valides dans CommandWords ainsi qu'un test dans la boucle principale du jeu qui, si cette commande était tapée renvoyait vers la méthode look(). Méthode look() qui utilise l'accesseur précédemment décrit : getLongDescription.

7.14.1 – look item :

Dans cet exercice le but était de créer une commande look avec l'ajout d'un second mot possible pour cibler la commande. Ainsi si la commande look est suivie d'un second mot, alors elle doit afficher seulement les informations concernant l'objet entré en second mot de la commande, si celui-ci existe bien. Pour ce faire il a fallu modifier les conditions d'analyse des commandes pour prendre en compte un cas si il y avait un deuxième mot dans la commande (hasSecondWord()) et si il était « égal » à l'item de la pièce en question, et un autre cas si il n'était pas « égal ». Dans les deux cas cela renvoie vers une procédure d'affichage de texte propre à chacune des conditions. Enfin si n'y a pas de second mot le programme renvoie vers la méthode regarde() classique.

7.15 – eat :

Cet exercice est identique dans la forme avec le précédent, en effet nous voulons afficher un texte prédéfini dans une méthode eat() quand l'utilisateur tape cette commande. Donc nous avons également rajouter la commande eat à l'ensemble des commandes valides ainsi qu'un test dans la boucle principale du jeu qui renvoie vers la méthode.

7.16 – showAll / showCommands :

En ajoutant précédemment les deux commandes, nous avons créé un couplage implicite. En effet lorsque le message du début ou celui du printHelp() s'affiche elles n'apparaissent pas. Ceci est dû au fait que nous n'avons pas créé d'accesseur dans la classe CommandWords pour accéder aux différentes commandes qui sont valides. Une fois cette chose faite il ne nous reste plus qu'à appeler cet accesseur quand cela est nécessaire.

7.17 – changer Game :

Si nous avons besoin de rajouter une commande, nous avons dorénavant plus besoin de modifier la classe Game. En effet il nous suffit de rajouter la commande à la liste des commandes valides dans la classe CommandWords si besoin de rajouter un test dans l'analyse des commandes entrées par l'utilisateur pour ajouter une fonction à cette commande mais noter que cette partie n'est pas nécessaire au bon fonctionnement du jeu.

7.18 – getCommandList :

Nous avons modifié le nom de la méthode showAll() en getCommandList().

7.18.1 – zuul-better :

J'ai comparé mon projet avec le projet zuul-better et je n'ai constaté aucun changement majeur mis à part l'ajout de la méthode getShortDescription() qui pour ma part n'ai pas appelée donc que je n'ai pas ajouté.

7.18.5 – HashMap aRooms :

Dans la même logique que le HashMap pour les sorties des Rooms, nous avons créé un HashMap pour les différentes Rooms qui seront donc plus faciles à récupérer juste en entrant la valeur de la « clé » correspondante.

7.18.6 – zuul-with-images :

J'ai comparé mon projet avec le projet zuul-with-images pour pouvoir ajouter une interface graphique ainsi que des images pour chaque pièce. De ce fait j'ai effectué les différents changements cités dans la consigne de l'exercice. De plus j'ai renommé toutes les variables et les attributs pour que cela soit correct vis-à-vis des conventions de nommage.

7.18.7 – addActionListener() / actionPerformed() :

La première méthode fait signifier à l'objet qu'il est « écouter » par UserInterface. La deuxième méthode est appelée à chaque fois qu'un évènement « sous écoute » de la classe en question se produit.

7.18.8 – Boutons :

Pour pouvoir ajouter un bouton j'ai dû rechercher sur internet quelle classe de l'interface graphique se chargeait de cela. J'ai trouvé que la classe en question est « JButton » et qu'il fallait de ce fait l'importer. Ensuite il a fallu créer ces boutons, y ajouter du texte et leur affecter une commande quand le bouton était pressé (dans la méthode actionPerformed).

7.19.2 – Répertoire Images :

J'ai bien ajouté toutes les images pour chaque pièce dans un répertoire *Images* ce qui va permettre un meilleur accès aux images après avoir modifié leur chemin d'accès dans la classe *GameEngine*.

7.20 / 7.21 – Item :

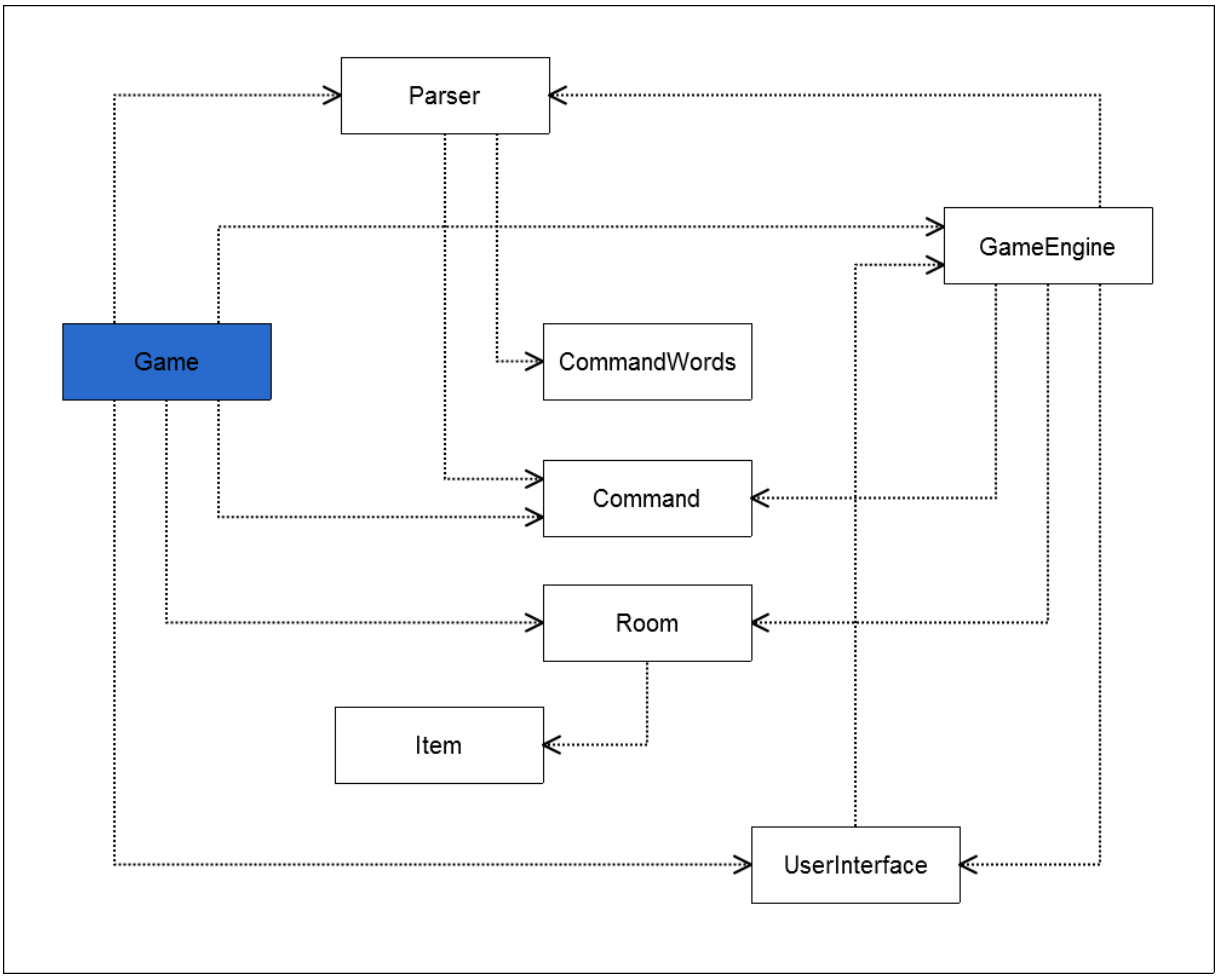
Dans cet exercice le but est de pouvoir ajouter un Item (objet) dans chaque pièce qui sera plus tard partie intégrante du jeu. Pour ce faire nous avons créé une classe *Item* ainsi que toutes les méthodes nécessaires :

- 2 attributs : *aDescription* & *aPoids* ;
- Un constructeur naturel pour l'initialisation de ces attributs ;
- 2 accesseurs pour chacun des attributs ;

Une fois cette classe créée nous avons ajouté un attribut dans chaque pièce : *aItem* ; qui dans le constructeur naturel est initialisé à *null* pour les cas où il n'y a pas d'item. Pour ajouter un item nous avons créé un modificateur : *setItem* qu'il ne reste plus qu'à appeler lors de la création des pièces. Enfin pour savoir s'il y a un objet dans une pièce il fallait mettre à jour la méthode *getLongDescription*.

ANNEXES :

7.12 :


Parser
GameEngine
Game
CommandWords
Command
Room
Item
UserInterface