

Rapport d'A3P

fillMyClass()

I.	Première partie	
	A.	Auteurs
	B.	Thème
	C.	Résumé du scénario
	D.	Plan
	E.	Détail des lieux, items, personnages
	F.	Situations gagnantes et perdantes
II.	Réponses aux exercices	
	1.	Exercice 7.0
	2.	Exercice 7.1
	3.	Exercice 7.1.1
	4.	Exercice 7.2
	5.	Exercice 7.2.1
	6.	Exercice 7.3
	7.	Exercice 7.3.1
	8.	Exercice 7.3.2
	9.	Exercice 7.4
	10.	Exercice 7.5 printLocationInfo()
	11.	Exercice 7.6 getExit()
	12.	Exercice 7.7 getExitsString()
	13.	Exercice 7.7.1
	14.	Exercice 7.8 HashMap, setExit()
	15.	Exercice 7.8.1
	16.	Exercice 7.9 keySet()
	17.	Exercice 7.10 getExitString()
	18.	Exercice 7.10.1
	19.	Exercice 7.10.2
	20.	Exercice 7.11 getLongDescription()
	21.	Exercice 7.14 look()
	22.	Exercice 7.15 eat()
	23.	Exercice 7.16 showAll(), showCommands()
	24.	Exercice 7.18 getCommandList()
	25.	Exercice 7.18.1
	26.	Exercice 7.18.3
	27.	Exercice 7.18.4
	28.	Exercice 7.18.6
	29.	Exercice 7.18.7
	30.	Exercice 7.18.8
	31.	Exercice 7.19.2
	32.	Exercice 7.20
	33.	Exercice 7.21
	34.	Exercice 7.22
	35.	Exercice 7.22.1
	36.	Exercice 7.22.2
	37.	Exercice 7.23 back()
	38.	Exercice 7.26
	39.	Exercice 7.26.1
	40.	Exercice 7.28.1
	41.	Exercice 7.28.2
	42.	Exercice 7.29
	43.	Exercice 7.30
	44.	Exercice 7.31
	45.	Exercice 7.31.1
	46.	Exercice 7.32
	47.	Exercice 7.33
	48.	Exercice 7.34
	49.	Exercice 7.34.1
	50.	Exercice 7.34.2
	51.	Exercice 7.35
	52.	Exercice 7.35.1
	53.	Exercice 7.40
	54.	Exercice 7.41
	55.	Exercice 7.41.1
	56.	Exercice 7.42
	57.	Exercice 7.42.2
	58.	Exercice 7.43
	59.	Exercice 7.44
	60.	Exercice 7.45.1
	61.	Exercice 7.45.2
	62.	Exercice 7.46
	63.	Exercice 7.46.1
	64.	Exercice 7.46.2
	65.	Exercice 7.46.3
	66.	Exercice 7.46.4
	67.	Exercice 7.47
	68.	Exercice 7.47.1
	69.	Exercice 7.47.2
III.	Mode d'emploi	
IV.	Déclaration anti-plagiat	

I. Première partie

A. Auteurs

Wallerand Delevacq, élève de la promotion 2020 ESIEE Paris. Passionné d'informatique. Connaissances en développement Web, HTML, CSS, PHP, JS, SQL, Java Android.

Directeur des systèmes informatique de l'association Esieespace.

Développeur de l'application android officielle de la liste BDE Hélios

Responsable Web du club Fonzy.

B. Thème

You need to write your own Class !

Vous êtes dans la peau d'un étudiant de première année qui apprend à développer en Java, votre but est de retrouver les différentes commandes à écrire pour compléter la class qui vous permettra de passer au niveau suivant.

C. Résumé du scénario

Comment gagner ?

Pour pouvoir gagner une partie de ce jeu, vous allez devoir retrouver les différentes commandes qui vont composer la class que vous allez devoir créer pour compléter le niveau. Ces commandes seront soit des mots soit des commandes simples. Les différents niveaux seront composés de salles représentant les endroits où l'on peut réellement trouver de l'aide (icampus, forum, internet, livre, etc.) chaque niveau sera un étage auquel on peut passer en entrant la bonne commande.

D. Plan

E. Détail des lieux, items, personnages

Les différents lieux sont donc tout d'abord la salle de cours (Classroom), Icampus, Internet, le livre (Book), le Forum. La pièce de " l'étage supérieur " est le niveau suivant (le lieu qui servira à la validation des items récupérés).

Les items sont le Clavier, le Cours, le Compileur, Bluej, la Methode, la Variable. C'est différents items sont nécessaires pour créer la Class que nous devons compléter.

L'unique personnage est l'élève qui est animé par le joueur.

F. Situations gagnantes et perdantes

La situation gagnante est lorsque le joueurs a récupéré tous les items et les a déposé dans la pièce du niveau supérieur.

La situation perdante est lorsque le joueurs n'a pu récupérer tous les items et les déposer dans la pièce supérieur. (lorsque par exemple il n'aura pas obtenu l'item en répondant mal à une énigme).

II. Réponses aux exercices

Exercice 7.0

Le premier exercice a consisté à créer la page web de notre projet sur notre page perso. l'adresse est <http://perso.esiee.fr/~delevacw>

Exercice 7.1

introduction aux différentes notions et termes du Java : Cohésion, couplage, réingénierie, Objet, Class, etc.

Exercice 7.1.1

Choix du thème du jeu : Un étudiant devra compléter une Class en ramenant les bons items pour passer au niveau suivant.

Exercice 7.2

Exercice 7.2.1

Découverte de la classe Parser et de l'objet Scanner afin de pouvoir saisir des informations au clavier.

Exercice 7.3

Conception du scénario du jeu (Voir section [Résumé du scénario](#)).

Exercice 7.3.1

Conception de ce rapport sur le modèle décrit par Denis Bureau.

Exercice 7.3.2

Réalisation du plan du jeu (Voir section [Plan](#)).

Exercice 7.4

Introduction au couplage et à la cohésion.

Exercice 7.5 printLocationInfo()

On crée une méthode printLocationInfo() dans la classe Game afin d'éviter une duplication de code. En effet, ces informations sont nécessaires dans la méthode printWelcome() mais aussi dans la méthode goRoom().

Exercice 7.6 getExit()

On ajoute une méthode getExit() dans la classe Room afin de pouvoir récupérer les différentes sorties possibles dans une Room.

Exercice 7.7 getExitsString()

On modifie la méthode printLocationInfo() de la classe Game puis on ajoute une fonction getExitsString() dans la classe Room, elle renvoie les sorties au lieu de les afficher.

Exercice 7.7.1

Mise à jour du rapport.

Exercice 7.8 HashMap, setExit()

Utilisation d'une HashMap afin de facilement stocker les sorties d'une Room. La méthode setExit() permet d'ajouter une sortie à une Room.

Exercice 7.8.1

Nous pouvons maintenant ajouter dans le scénario un déplacement vertical vers la salle du niveau suivant.

Exercice 7.9 keySet()

La méthode keySet() de la classe HashMap nous permet de récupérer toutes les clefs d'un tableau associatif. Une HashMap étant un tableau associatif.

Exercice 7.10 getExitString()

```
public String getExitString()
{
    String returnString = "Sorties :";
    Set<String> keys = exits.keySet();
    for(String exit : keys){
        returnString += ' ' + exit;
    }
    return returnString;
}
```

La méthode `getExitString()` initialise la String de retour avec la valeur "Sorties :". Elle appelle ensuite la fonction `keySet()` pour mettre toutes les clés des sorties de la pièce courante dans une variable `keys`. Set est un ensemble où chaque élément est unique. `keys` est ensuite utilisé dans une boucle `foreach`. Ainsi on associe à chaque itération de `keys` la String correspondante que l'on stocke dans `exit`. À chaque boucle, on ajoute `exit` à `returnString`.

Lorsque chaque clé a été examinée, on sort de la boucle et on fini par renvoyer la variable `returnString` qui comporte maintenant toutes les sorties formatées.

Exercice 7.10.1

Complétion de la javadoc.

Exercice 7.10.2

Génération de la javadoc.

Exercice 7.11 `getLongDescription()`

On ajoute la méthode `getLongDescription()` à la classe `Room` puis on modifie la méthode `printLocationInfo()`.

Exercice 7.14 `look()`

Cette méthode permet de ré-afficher les informations de la pièce courante. On ajoute cette commande à la classe `CommandWords`.

Exercice 7.15 `eat()`

Cette méthode de la classe `Game` affiche une simple phrase. On ajoute le mot de commande dans `CommandWords`.

Exercice 7.16 `showAll()`, `showCommands()`

Résolution de problème de couplage implicite avec la méthode `help()` qui n'affiche pas automatiquement les nouvelles commandes créées. On crée la méthode `showCommands()` dans la classe `Game` qui permet d'appeler la méthode `showAll()` de la classe `CommandWords`. Cette dernière affiche toutes les commandes valides disponibles.

Exercice 7.18 `getCommandList()`

La méthode `getCommandList()` permet à la classe `CommandWords` de produire la liste des commandes au lieu de l'afficher.

La méthode `showAll()` renvoie directement la String des commandes disponibles. Cette String est affichée dans la méthode `printHelp()` de la classe `Game`.

Exercice 7.18.1

Le projet correspond bien au projet Zuul-Better.

Exercice 7.18.3

Recherche et création des images du jeu.

Exercice 7.18.4

Choix du titre du jeu : `fillMyClass()`

Exercice 7.18.6

Étude du projet Zuul-with-images et modification de `fillMyClass()` afin d'utiliser les images.

Exercice 7.18.7

Exercice 7.18.8

Ajout d'un bouton en `borderlayout.EAST`.

Inclusion d'un `gridlayout` dans le `borderlayout.EAST` afin de pouvoir placer les boutons `help`, `look`, `eat`, `quit`.

Exercice 7.19.2

Création du dossier `./images/` afin d'y déplacer toutes les images du jeu.

Exercice 7.20

Création d'une nouvelle classe `Item` comportant deux attributs (`aWeight`, `aDescription`) avec leurs `getters` & `setters` respectifs.

Exercice 7.21

Création d'une méthode permettant d'obtenir la description d'un item. C'est la classe `GameEngine` qui se chargera de l'afficher.

Exercice 7.22

Ajout d'une `HashMap` dans la classe `Room` afin de contenir les items de la pièce.

Exercice 7.22.1

La `HashMap` est un choix judicieux car elle permet d'associer une clé à une valeur ainsi on associe un nom cours à l'item de notre choix et on peut donc facilement le récupérer.

Exercice 7.22.2

Ajout des items dans les différentes pièces du jeu.

Exercice 7.23 `back()`

Ajout de la méthode `back()` à `GameEngine` ainsi que d'un attribut `aPreviousRoom` afin de conserver la pièce précédente.

Exercice 7.26

On crée une `Stack` qui stocke progressivement les différentes `Room` et qui permet donc d'effectuer des `back()` successifs.

Exercice 7.26.1

Génération des deux javadoc.

Exercice 7.28.1

Création d'une commande test qui va parcourir chaque ligne d'un fichier `.txt` et exécuter les commandes correspondantes.

Exercice 7.28.2

Création de deux fichiers de commandes test répondant à différents scénarios.

Exercice 7.29

Création de la classe Player. Un objet Player comporte donc sa salle courante, les salles visitées précédemment, le poids maximum transportable, les items portés.

Exercice 7.30

Création des méthodes pickUpItem() et dropItem() afin de pouvoir récupérer un objet dans une pièce.

Exercice 7.31

Déjà traité car utilisation d'une HashMap dès le début de l'exercice 7.29 en prévision de ce scénario.

Exercice 7.31.1

Création de la classe ItemList afin de simplifier l'utilisation des listes d'items.

Exercice 7.32

Ajout d'un attribut qui permet de stocker un nombre limité d'objets et des méthodes en conséquence afin de calculer si l'on a assez de place pour prendre un objet.

Exercice 7.33

Création de la commande "inventory" qui permet d'afficher l'inventaire du joueur.

Exercice 7.34

Création du MagicCookie qui permet de doubler la capacité de l'inventaire quand il est mangé.

Exercice 7.34.1

Mise à jour des fichiers de test.

Exercice 7.34.2

Mise à jour des javadocs.

Exercice 7.35

Intégration des enum dans le projet, modification des classes existantes pour l'adapter à cette nouvelle configuration.

Exercice 7.35.1

Remplacement des if, else if par des switch.

Exercice 7.40

Étude des sources de zuul-with-enums-v2.jar et réalisation de l'exercice.

Exercice 7.41

Changement du mot associé à la commande help et observation de la réflexion automatique dans le jeu.

Exercice 7.41.1

Merge de zuul-with-enums-v2.jar et fillMyClass().jar

Exercice 7.42

Ajout d'une limite de temps sous forme de décrémentation d'un compteur à chaque changement de pièce.

Exercice 7.42.2

Étude de la pertinence d'un changement d'IHM complexe. Je remet cet exercice à plus tard afin de me concentrer sur le reste du projet. J'ai également découvert une librairie pour les IHM, javaFX. Cette librairie semble être intéressante et plus moderne que Swing. Je pense l'utiliser par la suite l'implémentation se rapprochant de celle des View d'android, un portage du jeu pourrait être envisagé.

Exercice 7.43

Suppression de la porte Forum → Icampus afin de réaliser l'exercice TrapDoor. Seul le chemin Icampus → Forum reste possible. Modification de la commande back afin de tester si `aLastRoom.peek() == null`. Modification de `goRoom()` afin d'ajouter une room null à `aLastRoom` si c'est une TrapDoor.

Exercice 7.44

Mise en place du Beamer : création de la sous classe d'Item Beamer. Création des méthodes fire et charge dans player.

Exercice 7.45.1

Modification des fichiers de test afin de tester le Beamer.

Exercice 7.45.2

Regénération des fichiers JavaDoc.

Exercice 7.46

Création d'une classe RoomRandomizer qui renvoi une Room au hasard. Création d'une TransporterRoom qui utilise RoomRandomizer.

Exercice 7.46.1

La commande aléa est fonctionnelle et permet de tester le jeu avec la TransporterRoom.

Exercice 7.46.2

Le choix de l'héritage avait déjà été fait auparavant.

Exercice 7.46.3

Ajout de certains commentaire javadoc dans le projet.

Exercice 7.46.4

Génération des deux javadocs pour le projet.

Exercice 7.47

Mise en place du système de Command avec héritage. Cela permet à présent de créer une nouvelle commande directement dans une classe Java. De plus la méthode interpretCommand de GameEngine est simplifiée.

Exercice 7.47.1

Création de package pour le projet :

- pkg_commands
- pkg_game
- pkg_item
- pkg_player
- pkg_room

Exercice 7.47.2

Génération des javadocs en prenant en compte les packages.

III. Mode d'emploi

Lancer directement le .jar (une méthode Main a été ajouté afin que ce dernier soit exécutable).

IV. Déclaration anti-plagiat

Aucun code autre que ceux fournis n'a été utilisé.