EXERCICE COMPLET

Le code auquel fait référence ce document est à récupérer en ligne, il se compose de deux fichiers. Le premier fichier, RPS.java : http://rdorat.free.fr/Enseignement/POO/Java/ExRef/RPS.java est le fichier principal, le fichier Lire.java : http://rdorat.free.fr/Enseignement/POO/JavaExRef/Lire.java ne sera pas étudié en tant que tel mais il nécessaire au fonctionnement de l'application. Ces deux fichiers sont à placer dans le répertoire bin de votre jdk. Le fichier Lire.java contient la définition de la classe public Lire qui récupère des données saisies par l'utilisateur au clavier.

Le fichier RPS.java contient la classe public RPS, la classe Partie, la classe JoueurHumain et la classe JoueurArtificel. Ces classes constituent une implémentation du jeu papier cailloux ciseaux (RPS pour rock, paper, scissors en anglais) : à chaque exécution de la classe RPS, une partie est lancée, l'utilisateur humain joue contre la machine. Pour tester ce code, il convient d'abord de le compiler. **javac RPS.java** compilera à la fois le fichier RPS.java et le fichier Lire.java. Une fois la compilation exécutée avec succès, il suffit d'exécuter le code : **java RPS**.

Avant de lire le document, vous êtres invité à lire le code RPS.java en tentant de le comprendre, à partir du cours et de ce que vous observez à l'exécution. Dans le présent document, on vous propose :

1. une description du fonctionnement du code. Cette première description se situe à un niveau général.
2. une description du fonctionnement en mémoire vive. Cette deuxième description sera très précise quand aux variables stockées en mémoire vive et aux valeurs qu'elles stockent. Comme ce type de description est très détaillée, elle ne portera pas sur l'entièreté du code.
3. Une série de questions auxquelles vous êtes invité à répondre, allant de simples questions de compréhension du code à des questions vous invitent à modifier le code existant pour l'améliorer.

I- Classes, comportements et fonctionnement du programme

Le jeu repose principalement sur les classes contenues dans RPS.java, on décrit leur contenu :

La classe JoueurArtificiel

Quand cette classe est instanciée, un nouveau joueurArtificiel est crée pour être confronté à un utilisateur humain

Les attributs :

- int score : un attribut de type int qui enregistre le score du joueurArtificiel

Les fonctions-méthodes :

- char getCoup(Random alea) : cette fonction prend un objet de type Random en paramètre, ce type d'objet permet de générer des valeurs aléatoire. Elle est appellée à chaque fois que le joueurArtificiel doit prendre une décision, c'est à dire choisir entre jouer pierre-cailloux ou ciseaux. La fonction tire au hasard un entier dans [0:2] et retourne 's' si elle a tiré 0, 'r' si elle a tiré 1, 'p' sinon.

La classe JoueurHumain

Quand cette classe est instanciée, un nouveau joueurHumain est crée.

Les attributs :

- int score : un attribut de type int qui enregistre le score du joueurHumain

Les fonctions-méthodes :

- char getCoup() : cette fonction est appelée à chaque fois que le joueurHumain doit prendre une décision, c'est à dire choisir entre jouer pierre-cailloux ou ciseaux. La fonction demande à l'utilisateur de rentrer 'r', 's' ou 'p' au clavier et lui redemande cela tant qu'il saisi une autre valeur. La fonction renvoie ce que l'utilisateur a saisi au clavier.

La classe Partie

Cette classe représente une partie jouée entre deux joueurs.

Les attributs :

- Random alea : un objet de type Random qui permet de générer des valeurs aléatoires.
- int nbCoups : un attribut de type int , le nombre de coups de la partie.
- JoueurHumain joueur1 : une partie se joue entre un utilisateur et un offreur artificiel. joueur1 représente le joueur humain.
- JoueurArtificiel joueur2 : une partie se joue entre un utilisateur et un offreur artificiel. joueur2 représente le joueur artificiel.

Les fonctions-méthodes :

- void generate() : cette fonction exécute effectivement la partie. Elle crée les objets alea, joueurHumain et joueurArtificiel. Puis elle entre dans une boucle pour laquelle chaque tour de boucle correspond à un tour de jeu. A chaque tour de boucle, on récupère le coup joué par l'offreur humain en appelant la fonction **joueur1.getCoup()**, on récupère le coup joué par l'offreur artificiel en appelant la fonction **joueur2.getCoup(alea)**, en fonction des coups joués, on fait évoluer les scores de chacun des joueurs. Par exemple, si le coup de l'ordinateur est 'r' et le coup de l'utilisateur est 'r', aucun ne marque. Si le coup de l'ordinateur est 'p' (Paper) et le coup de l'utilisateur 'r' (Rock), le joueur Artificiel marque un point (**joueur2.score++)**, le joueur humain n'en marque pas. A la fin du jeu, on affiche le vainqueur en comparant **joueur1.score** et **joueur2.score**.
- String getCoup(char c) : cette fonction renvoie le nom du coup en fonction du caractère défini par convention. En effet, plus haut, on a dit que 'r' correspond à pierre, 'p' à papier et 's' à ciseaux. Cette fonction renvoie une chaîne de caractères correspondant au nom du coup en fonction de la lettre associée.

La classe RPS

Cette classe ne contient qu'une fonction : le main qui lance tout le reste du programme. C'est cette classe qui est appellée lors de l'exécution (**java RPS**) puisque qu'elle contient le main.

Les fonctions-méthodes :

- main : elle demande à l'utilisateur si il veut joueur. Si ce dernier saisi 'o', la fonction crée une instance de Partie (**Partie p=new Partie()**) et appelle ensuite le déroulement de la partie (**p.generate()**).

Dans le déroulement d'une partie, une fois que l'utilisateur a saisi **java RPS** en invite de commandes, le programme récupère le contenu du fichier RPS.class, il y trouve la fonction main qu'il exécute. Cette fonction commence par demander à l'utilisateur de rentrer une valeur au clavier : 'o' si il veut joueur, 'n' sinon. Si l'utilisateur rentre 'o', une partie est créée (**Partie p=new Partie()**) puis lancée (**p.generate()**). Ce sont alors les instructions de **p.generate()** qui sont exécutées : on crée un objet de type Random, on crée un objet de type joueurHumain et un objet de type joueurArtificiel puis on joue les différents tours de jeu (la boucle). Dans un tour de jeu, on "demande " son coup à l'objet joueurHuamin (**joueur1.getCoup()**), on "demande" son coup à l'objet joueurArtificiel (**joueur2.getCoup(alea)**), on compare les coups pour savoir si l'un a remporté ce tour. On actualise les scores en fonction. A la fin des tours de jeu, on affiche le résultat découlant de la comparaison des scores. On sort alors de la fonction generate(). On sort ensuite de la fonction main, le programme est terminé.

II- Déroulement en mémoire vive

Il n'est pas absolument nécessaire de lire cette partie si la description de la partie précédente vous convient dans la compréhension du fonctionnement du code. La lecture de cette partie n'est même pas recommandée en première approche.

On décrit ici ce qui se passe instruction par instruction pour deux fonctions (la fonction main et la fonction getCoup de joueurArtificiel) en montrant l'impact en la mémoire vive de chaque instruction.

Le code est ici dépendant de l'interaction avec l'utilisateur qui va rentrer des données en cours d'exécution. Le déroulement du programme pourra donc varier en fonction des actions de l'utilisateur. Les choix de l'utilisateur sont notés en rouge. Par exemple : l'**utilisateur rentre 'o' au clavier**.

La fonction main

Au début du programme, c'est la fonction main(String args[]) de la classe RPS qui est appellée. On suit l'exécution de cette fonction instruction par instruction :

System.out.println("Voulez-vous jouer une partie ? o/n");

Cette instruction affiche

Voulez-vous jouer une partie ? o/n

à l'écran

char c=Lire.c();

Cette instruction crée une variable c de type char en mémoire vive. La fonction Lire.c() récupère l'entrée clavier de l'utilisateur et l'affecte à c. **On suppose que l''utilisateur rentre 'o' au clavier**

Les variables de la fonction main et leur gestion en mémoire

Nom	Type	octets occupés	Valeur enregistrée
c	char	100-107	'o'

```
if(c=='o')
{
Partie p=new Partie();
}
```

Si la variable c contient le caractère 'o', le programme crée un objet de type Partie en mémoire vive et affecte sa référence à p. Un objet Partie est composé : d'un attribut de type int (nbCoups), de trois variables objets (Random alea, joueur1, joueur2), soit 4*4=16 octets pour coder un objet de type Partie. On suppose qu'à la création la machine crée cet objet des octets 120 à 135.

Les variables de la fonction main et leur gestion en mémoire

Nom	Type	octets occupés	Valeur enregistrée
c	char	100-107	'o'
p	Partie	140-143	120
p.alea	Random	120-123	null
p.nbCoups	int	124-127	0
p.joueur1	JoueurHumain	128-131	null
p.joueur2	JoueurArtificiel	132-135	null

La fonction main continue en appelant la fonction p.generate(). Ce sont donc les instructions de la méthode generate() de Partie qui sont exécutées avec les valeurs enregistrées pour l'objet p. Quand on appelle p.generate(), on appelle une fonction sans paramètre. **Néanmoins, l'appel de fonction sur un objet correspond toujours au passage implicite d'une valeur qui est la référence de l'objet pour lequel on appelle la fonction**. La valeur de la variable p qui est l'adresse de l'objet Partie (120) pour lequel on appelle generate() est passée en paramètre : dans la fonction generate(), on dispose de this qui est un pointeur sur objet et qui contient l'adresse 120. Ainsi, les différents attributs de l'objet seront accessibles : this.alea, this.nbCoups etc... Au début de l'exécution de la fonction generate(), les variables visibles en mémoire sont dans cet état :

Les variables au début de la fonction generate et leur gestion en mémoire

Nom	Type	octets occupés	Valeur enregistrée
this	Partie	140-143	120
this.alea	Random	120-123	null
this.nbCoups	int	124-127	0
this.joueur1	JoueurHumain	128-131	null
this.joueur2	JoueurArtificiel	132-135	null

Notez que dans la fonction generate, il n'est pas nécessaire d'employer this.nbCoups ou this.joueur1, on peut se contenter de manipuler nbCoups, joueur1 : la machine cherchera d'abord parmi les variables locales et celles passées en paramètre puis elle ira voir dans les variables de l'objet à l'adresse this.

Après l'exécution de la fonction generate(), les variables associées à l'objet à l'adresse 120 ont été modifiées, dans main, l'état des

variables est alors :

Les variables de la fonction main et leur gestion en mémoire

Nom	Type	octets occupés	Valeur enregistrée
c	char	100-107	'o'
p	Partie	140-143	120
p.alea	Random	120-123	220
p.nbCoups	int	124-127	10
p.joueur1	JoueurHumain	128-131	310
p.joueur2	JoueurArtificiel	132-135	340

il n'y a plus d'instruction dans la fonction main après p.generate(), il n'y a pas de fenêtre affichée : l'exécution du programme s'arrête et les espaces mémoires sont libérés.

La fonction getCoup(Random alea) de JoueurArtificiel

L'appel à la fonction se fait à travers l'instruction joueur2.getCoup(alea); dans la fonction generate() de Partie. L'ojet alea passé en paramètre est de type Random. Dans la fonction generate(), l'objet alea a été préalablement instanciée, de même pour l'objet joueur2. Au moment de l'appel de joueur2.getCoup(alea), parmi les variables courantes pour la fonction generate, on a :

Les variables de la fonction main et leur gestion en mémoire

Nom	Type	octets occupés	Valeur enregistrée
alea	Random	120-123	220
joueur2	JoueurArtificiel	132-135	340

L'appel à **joueur2.getCoup(alea)** correspond à un appel où l'on passe en paramètre l'objet alea : en fait, la machine crée une nouvelle variable objet de type Random qui pointe sur le même objet en mémoire. De manière implicite, c'est le même mécanisme qui se produit pour le passage de la référence de l'objet : this, dans la fonction getCoup(alea) est une variable de type JoueurArtificiel qui a récupéré la valeur de joueur2 de generate(). Donc l'état de la mémoire est :

Impact en mémoire vive - les variables pour la fonction getCoup(alea) :

Nom	Type	octets occupés	Valeur enregistrée
alea	Random	410-413	220
this	JoueurArtificiel	440-443	340

La fonction s'exécute ensuite et renvoie un char.

III- Questions

1. Questions de compréhensions
    - Pourquoi le fichier s'appelle "RPS.java"
    - Quelles sont les classes utilisées en dehors de RPS, JoueurHumain, JoueurArtificiel et Partie ? Comment se fait il que la machine parviennent à les retrouver ?
    - Etablissez pour chaque classe utilisée la liste des méthodes et des attributs qui apparaissent dans ce code.
2. Comment modifier le programme pour que
    - Separer le programme en autant de fichiers qu'il y a de classes.
    - On voudrait que le nombre de coups soit choisi par l'utilisateur avant le jeu. Implementez cette solution.
    - En sachant que Lire.S() permet de récupérer une chaine de caractères String entrée au clavier, comment pourriez vous envisager de récupérer le nom de l'utilisateur et de le faire intervenir au moment de l'affichage des scores finaux.
    - Le code, tel qu'écrit actuellement, repose sur une convention : 'r' désigne le coup "pierre", 's' désigne le coup "ciseaux", 'p' désigne le coup "papier" : on se refère à la désignation anglaise du jeu RPS. Modifier le code de manière à ce que ce soit une convention française qui soit adoptée.
    - Actuellement, que se passe t'il si l'utilisateur saisi autre chose que 'o' ou 'n' lorsque la fonction Lire.c() de main est active ? Implémentez une solution pour contraindre l'utilisateur à saisir 'o' ou 'n'.
3. Perspectives
    - Comment implémenter un autre joueurArtificiel qui joue différemment de celui qui est actuellement proposé ? Proposez une solution et testez là.
    - Proposez un moyen de faire jouer votre nouveau joueurArtificiel contre le joueurArtificiel proposé initialement sur un grand nombre de coups : lequel gagne ?