TEST JSEBASES N°1 durée : 4H

CHASSE AU TRESOR

DANS LES ILES "SNICKERS"

Au cours d'un de ces nombreux voyages, le capitaine Jack SPAROW avait trouvé un trésor sur une des îles SNICKERS, non loin du piton sucré. Il en avait fait établir une carte qu'il détenait précieusement sous son bandeau. Hélas, trop occupé à sauver sa peau, il laissa choir son couvre-chef et perdit ainsi la fameuse carte...

Quelques années plus tard, au cours d'une enquête, John Rambo et Chuck NORRIS retrouvèrent ladite carte, au fond d'un paquet de bonbons Haribo. Tous les deux posèrent des permissions afin de partir ensemble à la chasse de ce trésor !

Bref: le contexte est posé !

Votre travail consistera à écrire en Java une application qui simulera le déplacement des deux Joueurs sur la même carte.
Vous respecterez le modèle MVP vu en cours !
John RAMBO partira des coordonnées du coin en haut à gauche, soit en [0,0], alors que Chuck NORIS partira lui du coin en bas et droite de la carte, soit en [9,9].
Comme vous avez pu le deviner la Carte sera un objet qui portera une grille de dimension 10x10 constituée de caractères. Les deux joueurs parcourent à tour de rôle la carte, ligne par ligne.
Le premier qui trouve le trésor arrête la ChasseAuTrésor!

I) Les affichages attendus :

```
>>debut de CHASSE au trésor
La grille:
  ./.A....B....C....D....E....F....G....H....I....J..
00|  .  |  .  |  .  |  .  |  .  |  .  |  P  |  P  |  .  |  P  |
01|  .  |  P  |  .  |  .  |  .  |  .  |  .  |  P  |  .  |  .  |
02|  .  |  .  |  P  |  .  |  .  |  .  |  .  |  .  |  .  |  .  |
03|  .  |  .  |  .  |  .  |  .  |  .  |  .  |  .  |  .  |  P  |
04|  .  |  .  |  .  |  P  |  .  |  .  |  .  |  .  |  .  |  .  |
05|  .  |  .  |  .  |  .  |  .  |  P  |  P  |  .  |  .  |  T  |
06|  .  |  .  |  .  |  P  |  .  |  .  |  P  |  .  |  .  |  .  |
07|  .  |  .  |  .  |  .  |  P  |  .  |  P  |  .  |  .  |  .  |
08|  .  |  .  |  .  |  P  |  .  |  .  |  .  |  .  |  .  |  .  |
09|  .  |  .  |  .  |  P  |  P  |  .  |  .  |  .  |  P  |  .  |
  .\.............................................../
no passage: 1 ,[  John.RAMBO] fin exploration ligne: 00, ligne suivante sera: 01
no passage: 1 ,[Chuck.NORRIS] fin exploration ligne: 09, ligne suivante sera: 08
no passage: 2 ,[  John.RAMBO] fin exploration ligne: 01, ligne suivante sera: 02
no passage: 2 ,[Chuck.NORRIS] fin exploration ligne: 08, ligne suivante sera: 07
no passage: 3 ,[  John.RAMBO] fin exploration ligne: 02, ligne suivante sera: 03
no passage: 3 ,[Chuck.NORRIS] fin exploration ligne: 07, ligne suivante sera: 06
no passage: 4 ,[  John.RAMBO] fin exploration ligne: 03, ligne suivante sera: 04
no passage: 4 ,[Chuck.NORRIS] fin exploration ligne: 06, ligne suivante sera: 05
no passage: 5 ,[  John.RAMBO] fin exploration ligne: 04, ligne suivante sera: 05
no passage: 5 ,[Chuck.NORRIS] fin exploration ligne: 05, ligne suivante sera: 04

--Les Résultats sont :
/------------IDENTITE-+-PV.-+-PAS-++---------\
|           JOHN.RAMBO | 042 |  50 ||         |
|         CHUCK.NORRIS | 042 |  41 || GAGNANT |
\----------------------+-----+-----++---------/
```

Vous réaliserez les affichages dans la classe OutilsIhm du package ihm !
Afin de réaliser cet affichage, vous devrez absolument utiliser l'objet StringBuilder et ne pas faire de concaténation de chaînes.

1) Afficher le contenu de la carte

2) Afficher le no de passage, l'identité du joueur
   Et le message : « fin exploration ligne : *noLignedebut,* ligne suivante sera : *noLigneSuivante*

3) Afficher les résultats sous la forme d'un tableau formaté comme dans l'exemple.

   - Une colonne pour l'identité
   - Une colonne pour les points de vie
   - Une colonne pour le nombre de pas
   - Une dernière colonne indiquant si le joueur est le gagnant (a trouvé le trésor)

NB1 Le formatage du tableau sera réalisé à l'aide de la méthode String.format !!!

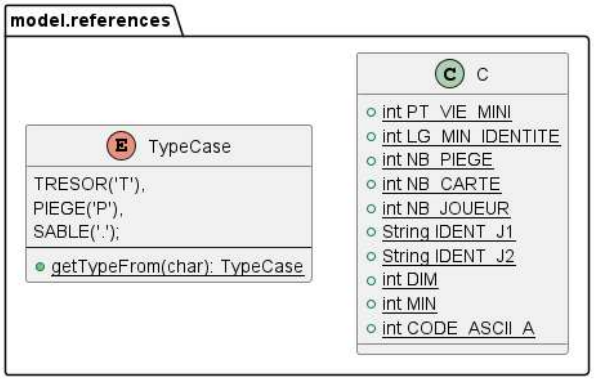
IMPORTANT : Il vous est interdit de reproduire uniquement avec du texte l'exemple proposé !!

II) les classe “métier” attendues :

- ***model.references***: les constantes attendues

La classe C servira à recenser toutes les constantes de l'application.

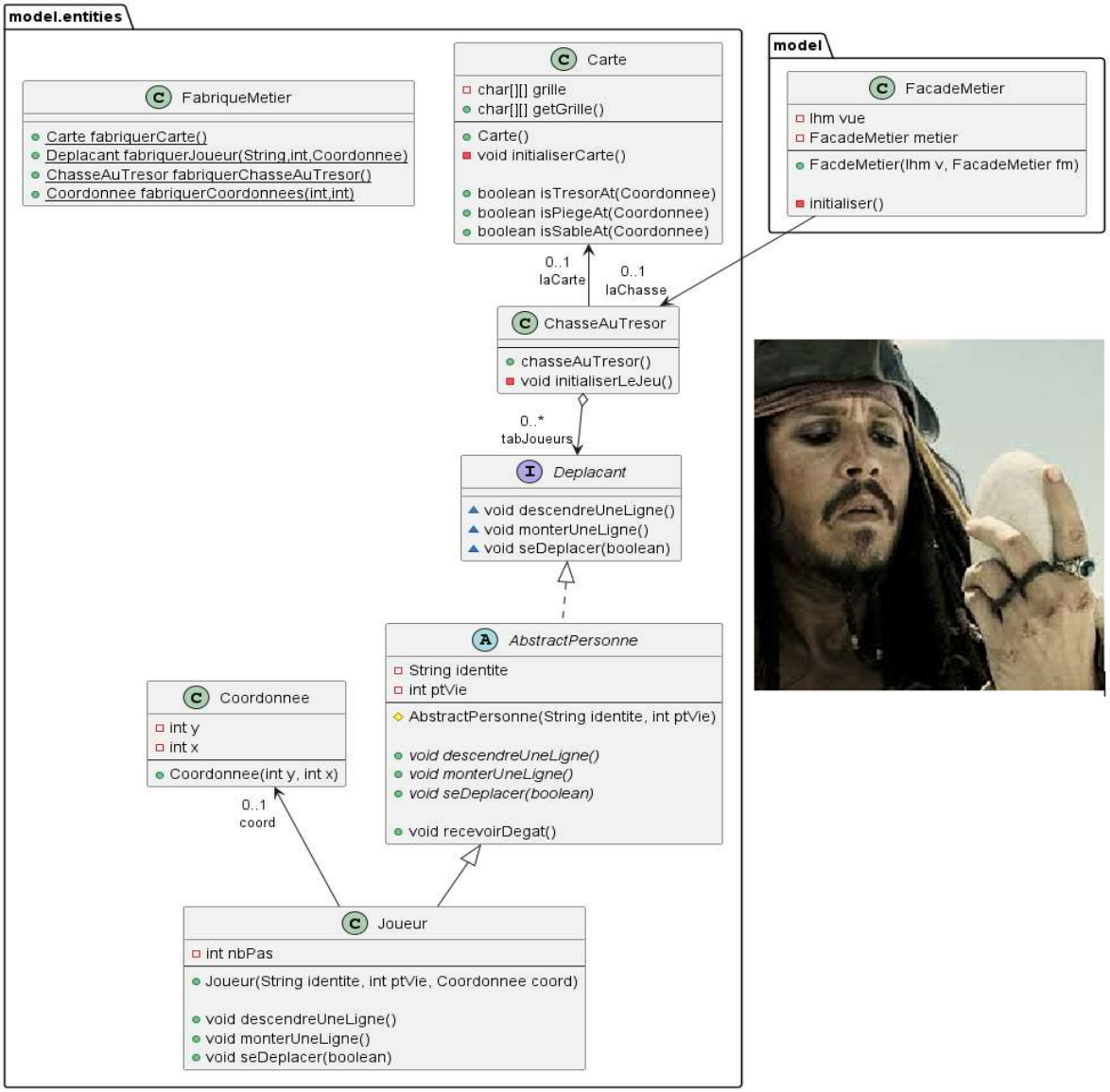
L'enum TypeCase contiendra les différentes valeurs possibles pour les cases de la carte.



Elle proposera à l’extérieur, sans devoir instancier la classe TypeCase, la méthode getType().
Le rôle de la méthode est de renvoyer le TypeCase correspondant au le char passé en paramètre, sinon renverra null.

NB : un seul Trésor et 10 pièges sont attendus

- ***model.entities***: les classes "métier"



II.A) une interface : Deplacant

Cette interface déclarera les méthodes suivantes :

```
+ void descendreUneLigne() :  qui augmentera le no de ligne de 1

+ void monterUneLigne() :          qui réduira le no de ligne de 1

+ void seDeplacer(boolean) :  qui gérera le déplacement :
      càd incrémentera le nombre de pas
      et infligera des dégâts si le boolean est à true
                  (cas du déplacement sur un piège !)
```

II.B) la classe ChasseAuTrésor: la classe principale de votre application.

Elle devra porter les attributs suivants :

-laCarte qui sera de type ***Carte***

-tabJoueurs qui sera un tableau de ***Deplacant*** /!\\ATTENTION

Elle possédera un constructeur vide qui utilisera la méthode ***initialiserLeJeu()*** de visibilité private et qui permettra de mettre en place les données de l'application et qui générera laCarte.

II.C) la classe Carte : elle portera comme unique attribut un tableau de caractère à deux dimensions de 10 X 10.

Elle possédera un constructeur vide qui utilisera la méthode ***initialiserCarte()*** de visibilité private pour générer le contenu de la carte.

Elle portera les méthodes suivantes :

```
+ boolean isTresorAt(Coordonnee) : méthode qui renvera true si le trésor
est placé à la coordonnée passée en paramètre

+ boolean isPiegeAt(Coordonnee) : méthode qui renvera true si un piège est
placé à la coordonnée passée en paramètre

+ boolean isSableAt(Coordonnee) : méthode qui renvera true si du sable est
placé à la coordonnée passée en paramètre
```

II.D) les classes AbstracPersonne et Joueur :

- ***AbstractPersonn***e sera une classe abstraite et portera les attributs suivants:

-identite de type chaîne : qui contiendra l'identité

-ptVie de type entier : qui stockera les points de vie

Le constructeur de cette classe sera de visibilité protected et n'attendra que le paramètre identité de type String.

NB : Les point de vie seront générés de façon aléatoire ; un minimum de 50 est attendu.

Elle contiendra les méthodes publiques suivantes :

```
+ {abstract}void descendreUneLigne() : à redéfinir dans une classe fille
+ {abstract}void monterUneLigne() : à redéfinir dans une classe fille
+ {abstract}void seDeplacer(boolean) : à redéfinir dans une classe fille

+ void recevoirDegat() : méthode sui est chargée de réduire les points de
vie de 1 point
```

- ***Joueu***r sera une classe normale qui sera constituée des attributs suivants :
    - -noj: de type entier, stockera le numéro du joueur
    - -nbPas de type entier, stockera le nombre de pas parcouru
    - -coords de typeCoordonnee: qui servira à stocker les coordonnées du joueur sur la carte lors de ses déplacements.

Cette classe sera une spécialisation de la classe AbstractPersonne (notion d'héritage) !
Le constructeur de cette classe attendra 3 paramètres : l'identité en String, le pv en entier, le numéro du joueur en entier.

Elle comportera les méthodes suivantes :

```
+ void descendreUneLigne() : qui incrémentera la coordonnee y (no d e ligne) de 1

+ void monterUneLigne() : qui décrémentera la coordonnée y (no de ligne) De 1

+ void seDeplacer(boolean) : qui incrémentera le nb de pas du jouer et qui infligera des dégâts si le boolean vaut true(cas d'un déplacement sur un piège)
```

NB: Les joueurs parcourent la même carte ligne par ligne à tour de rôle , seul le sens change.

Le joueur 0 commence en [y=0 :x= 0], parcourra la carte depuis la ligne 0 jusqu'à la ligne 9 en passant dans l'ordre des colonnes (de 0 à 9), puis passera à la ligne suivante au tour suivant.

Le joueur 1 qui commence en [y=9 :x= 9] parcourra la carte depuis la ligne 9 jusqu'à la ligne 0, en passant sur chaque colonne dans l'ordre inverse (de 9 à 0) et remontera d'une ligne au tour suivant.

II.E) la classe Coordonnées :
Cette classe portera les deux attributs :

- y de type entier : la ligne où se trouve le joueur
- x de type entier : la colonne où se trouve le joueur

Le constructeur de cette classe attendra les deux paramètres qui correspondent à ses attributs.

II.F) méthode attendues et non décrites :

Chaque classe devra posséder la méthode *toString() . Les méthodes equals() et hashCode() devront être générées quand c'est possible : pas pour la FacadeMetier, laFabriqueMetier ni l'interface.*
Vous devez OBLIGATOIREMENT passer par la FabriqueMetier pour instancier vos objets !!!

III) la facade Metier : sera nommée FacadeMetier et portera les données de l'application soit une ChasseAuTresor !
Son constructeur sera le constructeur vide qui fera appel à la méthode *private initialiser()* pour charger les données.

Elle possédera les 3 méthodes publiques suivantes :

```
+ Carte selectionnerCarte() : qui permettra de renvoyer l'objet Carte

+ char deplacerUnJoueur(int) : qui se chargera de déplacer le joueur
dont l'indice est passé en paramètre
```

IV) MVP, Lanceur,Presenter , etc...
Comme vu dans les TD et TP pratiqués, c'est le lanceur qui passera une Ihm et la FacadeMétier au Presenter, avant de lancer la méthode exec(). De plus, ce sera le dernier niveau pour faire quelque chose lors d'une Exception, il est interdit d'ajouter throws Exception dans la signature de la méthode main !!!

REMARQUE: le moteur de votre application devra se trouver dans la méthode ***exec()*** du Presenter.

V) BONUS: Vous n'êtes autorisé à coder le bonus que si vous avez fini le reste ! 2points
Vous devrez afficher les coordonnes du trésor après le tableau des résultats sous la forme :

```
Trésor trouvé en en J:6
```

NB : Avec J étant le no de colonne et 6 le numéro de la ligne !

BON COURAGE

CONSIGNES DE FIN DE TEST:
Votre projet devra se nommer : ***projetCAT_VOTRE_NOM*** !
Une fois le temps imparti vous livrerez votre code dans l'ECRIT_POUR de votre formateur (gilles.penaud)

Barème:

L'application est capable d'afficher ce qui est demandé:	Sur 20 points
L'application est capable d'afficher qqchose et ne plante pas: OU L'application n'affiche rien mais ne plante pas	Sur 15 points
L'application n'affiche rien et plante à l'exécution	Sur 12points
L'application ne se lance pas	Sur 10 points

MALUS (assuré ? avec ou sans franchise .?.)

Affichage par recopie textuelle de l'exemple	-4points
Pas/ Peu de commentaire javadoc	-1pt
Classe utilitaire/constantes: final et constructeur en private	-1pt
Utilisation de concaténation de chaîne au lieu StringBuilder	-2pt
Non utilisation de la FabriqueMetier pour instancier	-2pt
Non utilisation des Exceptions pour gérer les problèmes Notamment dans les setters	-2pt
Non respect des classes métiers	-1pt
Non respect du nommage d'une méthode : débute par un verbe à l'infinitif	-2pt