SaÉ 1.01 : ObiSoft

Contexte général

Vous poussez les portes vitrées de l'imposant bâtiment de ObiSoft Inc., le cœur battant d'excitation. À 20 ans, vous n'en revenez toujours pas d'avoir décroché un stage de trois mois dans cette entreprise légendaire, créatrice de certains des jeux vidéo les plus acclamés de la décennie. Les affiches géantes ornant le hall d'entrée témoignaient de leurs succès planétaires : "Odyssée Stellaire", "Chroniques d'Avalon", "Néo Tokyo 2077"... Autant de titres qui ont bercé votre adolescence et nourri votre passion pour la programmation. Alors que vous suivez votre tutrice, Hana, vers les bureaux de l'équipe de développement, vous découvrez avec émerveillement les *open spaces* modernes où s'affairent des centaines de talents créatifs. Les prochains mois allaient être intenses, parsemés de défis techniques stimulants et de petites victoires qui contribueront, peut-être, à façonner le prochain chef-d'œuvre d’ObiSoft Inc !

Première réunion avec Hana

Hana : « *Bienvenue ! Voici comment va se dérouler ton stage. Chaque semaine, tu devras écrire trois fonctionnalités différentes.* »
Vous : « *D'accord, ça me semble intéressant !* »
Hana : « *Ce n'est pas tout. À la fin de chaque semaine, je tirerai au hasard l'une de ces fonctionnalités.* »
Vous : « *Et ensuite ?* »
Hana : « *Tu auras alors 10 minutes pour réécrire cette fonctionnalité, mais sur papier cette fois.* »
Vous : « *Sur papier ? Vraiment ?* »
Hana : « *Exactement. C'est pour s'assurer que tu maîtrises bien ta solution, sans l'aide de l'ordinateur.* »
Vous : « *Je comprends. Et... est-ce que j'aurai le droit à des documents pendant ces 10 minutes ?* »
Hana : « *Non, ce sera uniquement toi, un stylo et une feuille de papier. L'objectif est de tester ta compréhension et ta mémoire.* »
Vous : « *Je vois. Ça va être un défi intéressant !* »
Hana : « *C'est le but ! Ça te permettra de vraiment intégrer ce que tu apprends. Prêt à commencer ?* »

Semaine 1 : Mise en route

Pour cette première semaine, différentes fonctionnalités concernant la création de personnages non joueur (PNJ) doivent être développées. Ces personnages ont différentes caractéristiques et il est nécessaire de pouvoir les créer facilement, tout en vérifiant que les données sont valides. Chaque caractéristique est définie par son nom et une valeur numérique comprise entre 0 et 10 inclus.

Fonctionnalité 1 : **Affichage des caractéristiques d'un personnage**

Un personnage est défini par différentes caractéristiques ayant une valeur numérique comprise entre 0 et 10. Pour l'instant, on ne doit saisir que trois caractéristiques (*Force*, *Agilité* et *Sagesse*), les deux autres (*Dextérité* et *Charisme*) étant automatiquement calculées. Ainsi, la *Dextérité* vaut la moyenne entre la *Force* et l'*Agilité* arrondie à l'inférieur. Tandis que le *Charisme* vaut 2 fois la *Force* plus la *Sagesse* le tout divisé par 3.

Écrivez le programme `AfficherStats` qui saisit successivement ces 3 informations et produit un affichage similaire à l'exemple de sortie donné ci-dessous.

```
> ijava AfficherStats
Force = 5
Agilité = 6
Sagesse = 5
Force (5) - Agilité (6) - Sagesse (5) - Dextérité (5) – Charisme (5)
```

Fonctionnalité 2 : **Saisie contrôlée d'une caractéristique**

Afin de créer un personnage, il est nécessaire de saisir la valeur d'une caractéristique donnée en vérifiant qu'elle est bien comprise entre 0 et 10.

Écrivez le programme `SaisirStat` qui saisit une caractéristique parmi les trois possibles, contrôle les données et produit des affichages similaires aux exemples de sortie donnés ci-dessous :

```
> ijava SaisirStat
Quelle caractéristique [Force, Agilité, Sagesse] ? force
Désolé cette caractéristique n'existe pas.

> ijava SaisirStat
Quelle caractéristique [Force, Agilité, Sagesse] ? Force
Quelle valeur [0-10] ? 20
Désolé cette valeur n'est pas valide.
```

```
> ijava SaisirStat
Quelle caractéristique [Force, Agilité, Sagesse] ? Agilité
Quelle valeur [0-10] ? 10
Agilité (10)
```

Fonctionnalité 3 : Génération aléatoire des caractéristiques d'un personnage

Afin d'automatiser la création des *Personnages Non Joueur* (PNJ), on souhaite générer les caractéristiques à l'aide de la fonction `double random()` qui retourne un réel compris dans l'intervalle [0.0, 1.0[.

Écrivez le programme `GenererPNJ` qui génère aléatoirement les caractéristiques d'un PNJ et produit des affichages similaires aux exemples de sortie donnés ci-dessous :

```
ijava GenererPNJ
Force (5) - Agilité (1) - Sagesse (8) – Dextérité (3) - Charisme (6)

ijava GenererPNJ
Force (3) - Agilité (10) - Sagesse (0) – Dextérité (6) - Charisme
(2)
```

Semaine 2 : Échauffement

Fonctionnalité 1 : Déterminer la réussite d'une action via un lancer de dé

On souhaite réaliser un des éléments du système de combat. Chaque attaque a un seuil de réussite qui est un nombre compris entre 0 et 10 inclus. On détermine la réussite de l'attaque par un jeter de dé : elle est réussie si, et seulement si, la valeur du dé est supérieure ou égale au seuil. De plus, on parle d'échec critique si la valeur du dé est 0, et de réussite critique si la valeur du dé est 10.

Vous devez écrire un programme qui demande le seuil de réussite de l'attaque, puis effectue un jeter d'un dé à 11 faces (dont les valeurs possibles sont entre 0 et 10 inclus) et affiche si l'action réussit ou échoue, et si c'est un échec critique ou une réussite critique.

Écrivez le programme `ActionReussie` qui évalue le niveau de réussite d'une action sur une caractéristique saisie manuellement, en fonction d'un lancer de dé.

```
ijava ActionReussie
Valeur de la caractéristique : 6
Résultat du dé : 4
Résultats de l'action : Échec

ijava ActionReussie
Valeur de la caractéristique : 5
Résultat du dé : 5
Résultats de l'action : Réussite

ijava ActionReussie
Valeur de la caractéristique : 7
Résultat du dé  0
Résultats de l'action : Échec critique !

ijava ActionReussie
Valeur de la caractéristique : 6
Résultat du dé : 10
Résultats de l'action : Réussite critique !
```

Fonctionnalité 2 : contrôle de saisie des caractéristiques d'un PNJ

On souhaite pouvoir créer manuellement un PNJ en saisissant les valeurs au clavier et en s'assurant de leur validité, c'est-à-dire qu'elles soient comprises entre 0 et 10 inclus.

Écrivez le programme `SaisirPNJ` qui réalise cette fonctionnalité et produit des affichages strictement identiques à ceux ci-dessous.

```
ijava SaisirPNJ
Veuillez entrer un nombre compris entre 0 et 10 inclus.
Force   : 5
Agilité : 10
Sagesse : 11
Veuillez entrer un nombre compris entre 0 et 10 inclus.
Sagesse : 7
Force (5) - Agilité (10) - Sagesse (7)
```

Fonctionnalité 3 : création d'un cadre contenant un message

Pour un des jeux imitant le mode console typique des années 70, Hana vous demande de créer un programme affichant un texte à l'intérieur d'un cadre dessiné à l'aide de caractères. De l'*ASCII Art* très primaire, mais c'est la demande du designer de ce jeu rétro ...

Écrivez le programme `CreerCadre` qui réalise cette fonctionnalité et produit des affichages strictement identiques à ceux ci-dessous.

```
ijava CreerCadre
Message : Hello World !
 ---------------
| Hello World ! |
 ---------------

ijava CreerCadre
Message : O
 ---
| O |
 ---

ijava CreerCadre
Message :
 --
|  |
 --
```

Semaine 3 : Entraînement renforcé

A partir de cette semaine, Hana vous demande de créer un seul programme `SaisieDePNJ` contenant une fonction pour chaque fonctionnalité, ainsi qu'un algorithme principal appelant ces fonctions afin de vérifier (manuellement) leur validité.

Fonctionnalité 1 : calcul d'un minimum

On souhaite disposer d'une fonction retournant le plus petit nombre entre deux entiers donnés en paramètre.

Écrivez la fonction `minimum` nécessitant deux paramètres entier et retournant le plus petit des deux et un algorithme principal la testant avec les valeurs permettant de produire des affichages strictement identiques à ceux ci-dessous.

```
ijava SaisieDePNJ
Dans (2, 3), le minimum est : 2
Dans (2, 0), le minimum est : 0
Dans (1, 1), le minimum est : 1
------------------------------
```

(notez qu'il n'y a aucune saisie dans la fonction et l'algorithme principal)

Fonctionnalité 2 : contrôle de saisie avec message

On souhaite pouvoir créer manuellement un PNJ en saisissant les valeurs des caractéristiques au clavier et en s'assurant de leur validité, c'est-à-dire qu'elles soient toujours comprises entre une borne minimale et une borne maximale toutes deux incluses.

Écrivez la fonction `saisir` nécessitant trois paramètres réalisant l'affichage d'un message et le contrôle de la valeur saisie afin qu'elle soit comprise dans l'intervalle [borneMinimale, borneMaximale], ainsi qu'un algorithme principal réalisant les saisies et produisant des affichages strictement identiques à ceux ci-dessous.

```
ijava SaisieDePNJ
Dans (2, 3), le minimum est : 2
Dans (2, 0), le minimum est : 0
Dans (1, 1), le minimum est : 1
------------------------------
Caractéristique : Force
Force : 17
Désolé, la valeur doit être dans l'intervalle [0,10].
Force : 10
Force vaut 10.
------------------------------
```

(on suppose dans l'exécution ci-dessus que le nom de la caractéristique est saisi dans l'algorithme principal et que la fonction est appelée avec 0 pour la borne minimale et 10 pour la borne maximale)

Fonctionnalité 3 : contrôle de saisie des caractéristiques d'un PNJ avec un nombre de points définis à répartir

On souhaite pouvoir créer manuellement un PNJ en saisissant les valeurs au clavier et en s'assurant de leur validité, c'est-à-dire qu'elles soient comprises entre 0 et 10 inclus, mais en ayant en plus une contrainte sur le nombre total de points à répartir sur les 3 caractéristiques. Il faut réutiliser la première fonctionnalité.

Écrivez la fonction `saisirPNJ` qui prend en paramètre le total de points à ne pas dépasser en cumulant les caractéristiques et retourne une chaîne de caractères contenant les caractéristiques et leur valeur, ainsi qu'un algorithme principal produisant des affichages strictement identiques à ceux ci-dessous.

```
ijava SaisirPNJ
Dans (2, 3), le minimum est : 2
Dans (2, 0), le minimum est : 0
Dans (1, 1), le minimum est : 1
------------------------------
Caractéristique : Force
Force : 10
Force vaut 10.
------------------------------
Nombre de points à répartir : 17
Force : 5
Agilité : 10
Sagesse : 4
Désolé la saisie doit être dans l'intervalle [0,2].
Sagesse : 2
Force (5) - Agilité (10) - Sagesse (2)
------------------------------

ijava SaisirPNJ
Dans (2, 3), le minimum est : 2
Dans (2, 0), le minimum est : 0
Dans (1, 1), le minimum est : 1
------------------------------
Caractéristique : Sagesse
Sagesse : 3
Sagesse vaut 3.
------------------------------
Nombre de points à répartir : 17
Force : 10
Agilité : 7
Il ne reste plus de points à répartir.
Force (10) - Agilité (7) - Sagesse (0)
------------------------------
```

(la saisie du nombre de point à répartir est réalisée dans l'algorithme principal, ensuite la fonction `saisirPNJ` est appelée et construit la chaîne de caractères apparaissant sur la dernière ligne de la trace d'exécution ci-dessus.)

Semaine 4 : Petits challenges

Fonctionnalité 1 : Toujours écrire d'abord le test d'une fonction avant de la définir !

Afin de visualiser les caractéristiques d'un personnage, Hana vous demande d'implémenter une première fonctionnalité capable de générer une représentation "graphique" d'un nombre.
Vous devez créer la fonction `genererBarre` qui répond aux tests suivants :

```
void testGenererBarre(){
  assertEquals("□□□□□□□□□□", genererBarre(0, 10));
  assertEquals("■■■□□□□□□□", genererBarre(3, 10));
  assertEquals("■■■■■■■■■■", genererBarre(10, 10));
}
```

Fonctionnalité 2 : Définir le code fonctionnel correspondant à un test existant.

A l'aide de la fonctionnalité développée précédemment, on souhaite maintenant une visualisation de ce type pour les caractéristiques d'un PNJ :

```
■■■□□□□□□□ Force
■■■□□□□□□□ Agilité
■■■■■■■■■■ Sagesse
```

Hana vous donne la signature de la fonction qui réalisera cette fonctionnalité :
`String visualiserCaracteristique(String nom, int valeur, int max)`

Afin de prendre de bonnes habitudes de développement, Hana vous demande d'écrire d'abord la fonction de test avant d'écrire le code fonctionnel pour la fonction `visualiserCaracteristique`.

Une fois que vous avez écrit cette fonction de test, intégrez la signature de la fonction `visualiserCaracteristique` et mettez juste `return ""` dans le corps de cette fonction.

Compilez et exécutez le programme, qui devrait normalement vous afficher le test en rouge. Seulement maintenant, écrivez le corps de la fonction `visualiserCaracteristique`.

Fonctionnalité 3 : Création automatisée d'un PNJ

Les scénaristes d'un jeu souhaitent pouvoir créer manuellement des personnages non joueur, mais les développeurs et développeuses préfèrent automatiser cette tâche. Le compromis a aboutit à ce que les scénaristes fournissent des descriptions de PNJ dans un fichier ayant cette forme :

```
Frodon,Force,4,Agilité,9,Sagesse,9
Z6PO,Force,1,Agilité,1,Langues,10
Beetlejuice,Magie,6,Malice,10,Sarcasme,10
```

Vous pouvez considérer que les descriptions transmises par les scénaristes sont correctement formatées (pas d'espace en plus, toujours une virgule pour séparer les différentes informations sur une ligne ...)

Hana vous demande d'écrire la fonction `algorithm` qui définit trois constantes contenant les exemples de personnages donnés ci-dessus et effectue l'appel d'une fonction sur chacune de ces constantes afin de produire exactement cet affichage :

```
Frodon
■■■■□□□□□□ Force
■■■■■■■■■□ Agilité
■■■■■■■■■□ Sagesse

Z6PO
■□□□□□□□□□ Force
■□□□□□□□□□ Agilité
■■■■■■■■■■ Langues

Beetlejuice
■■■■■■□□□□ Magie
■■■■■■■■■■ Malice
■■■■■■■■■■ Sarcasme
```

(Il peut être pratique d'avoir une fonction `String getChamps(String lignePNJ, int indice)` qui retourne le nième champs (le séparateur étant des virgules) pour récupérer l'information vous intéressant dans une ligne de PNJ . Ainsi getChamps("`Frodon,Force,4,Agilité,9,Sagesse,9`", 1) retournera "`Force`")

PS : vous pouvez aussi utiliser la fonction prédéfinie `int stringToInt( String s)`.

Semaine 5 : Challenges

Hana vous rappelle qu'il faut toujours créer un seul programme `Challenges` qui définit chaque fonctionnalité sous la forme d'une fonction (ou procédure ...).

Fonctionnalité 1 : Affichage formaté des caractéristiques d'un PNJ

On souhaite pouvoir afficher proprement les caractéristiques d'un PNJ sous la forme d'informations formatées. Le jeu étant disponible en plusieurs langues, il faut écrire un programme s'adaptant à la caractéristique au nom le plus long.

Écrivez la fonction `printPNJ` qui prend en paramètre 3 chaînes de caractères et 3 entiers (représentant le noms des caractéristiques et leurs valeurs) et crée une chaîne de caractères correspondant strictement aux affichages donnés ci-dessous se produisant lors de l'exécution de l'algorithme principal.

```
ijava Challenges
Strength (5)
Agility  (10)
Wisdom   (8)

ijava Challenges
Force   (5)
Agilité (10)
Sagesse (8)
```

(La première trace correspond à un appel avec les paramètres "Strength", 5, "Agility", 10, "Wisdom", 8 et la seconde aux paramètres "Force", 5, "Agilité", 10, "Sagesse", 8)

Hana vous fournit la fonction de test suivante pour vérifier votre travail. Le caractère `'\n'` correspond au retour à la ligne.

```
void testPrintPNJ() {
  assertEquals("a   (10)\naaa (7)\naa  (8)\n", printPNJ("a",10,"aaa",7,"aa", 8);
}
```

Fonctionnalité 2 : Affichage d'un PNJ défini par des tableaux

Une autre équipe travaille avec un format différent pour représenter les PNJ : deux tableaux, un de chaînes de caractères et l'autre d'entiers.

Ainsi, le classique Frodon, représenté la semaine passée par la chaîne de caractères :

```
"Frodon,Force,4,Agilité,9,Sagesse,9"
```

est transformé en un tableau de 4 chaînes de caractères contenant les chaînes (`"Frodon", "Force", "Agilité", "Sagesse"`) et un tableau de 4 entiers contenant les entiers (0,4,9,9).

Écrivez la fonction `printPNJ` qui prend en paramètre un tableau de chaînes de caractères et un tableau d'entiers et un algorithme principal qui produit un affichage strictement identique à celui ci-dessous.

```
ijava Challenges
Force   (5)
Agilité (10)
Sagesse (8)
----------------
Frodon
Force   (4)
Agilité (9)
Sagesse (9)
----------------
```

Hana vous fournit la fonction de test suivante :

```
void testPrintPNJTableaux() {
  assertEquals("Frodon\nForce   (4)\nAgilité (9)\nSagesse (9)\n", printPNJ(new
String[]{"Frodon", "Force", "Agilité", "Sagesse"}, new int[]{0,4,9,9}) ;
}
```

(il est judicieux de se rappeler ce que vous avez déjà écrit pour la fonctionnalité 1)

Fonctionnalité 3 : Création d'un niveau simplifié de démineur

Hana vous demande de réaliser une fonctionnalité nécessaire pour la création d'un nouveau jeu de type « démineur ». Dans ce jeu un niveau est un plateau à une dimension de booléens, indiquant si oui ou non une bombe est présente dans la case.
Hana vous indique qu'il est plus judicieux de d'abord placer toutes les bombes au début du tableau et ensuite de réaliser des permutations en tirant un indice aléatoirement dans le tableau.

Écrivez la fonction `generer` qui prend en paramètre un nombre de cases et un nombre de bombes et qui retourne un tableau de booléens contenant autant de cases à `true` que de bombes.
Pour tester votre fonction, il serait pratique de définir aussi une fonction println pour le type de tableau que vous retournez afin de l'afficher dans l'algorithme principal.

Il sera pratique de définir une fonction `String toString(boolean[] champs)` afin de générer une représentation similaire à ci-dessous du tableau créé par la fontion `generer`.

Votre affichage final lors de l'exécution du programme `Challenges` doit ressembler à cela :

```
ijava Challenges
Force   (5)
Agilité (10)
Sagesse (8)
---------------
Frodon
Force   (4)
Agilité (9)
Sagesse (9)
---------------
Nombre de cases  : 6
Nombre de bombes : 2
..B.B.
```