EXAMEN DE FIN D'ÉTUDES SECONDAIRES – Sessions 2024

QUESTIONNAIRE

Date :	05.06.24		*Horaire :*	14:15 - 17:15	*Durée :*	180 minutes
Discipline :	INFOR	*Type :*	écrit	*Section(s) :*	GIG	
				Numéro du candidat :		

Dans votre répertoire de travail (à définir par chaque lycée), vous trouverez un sous-dossier nommé **EXAMEN_GIG**. Renommez ce dossier en remplaçant le nom par votre numéro de candidat (exemple de notation : **LXY_GIG2_07**). Tous vos fichiers devront être sauvegardés à l'intérieur de ce sous-dossier, qui sera appelé **votre dossier** dans la suite !

Vous trouvez une version exécutable du jeu – **Demo.jar** – dans votre dossier.

Avant de commencer, il est recommandé de lancer et d'essayer ce jeu.

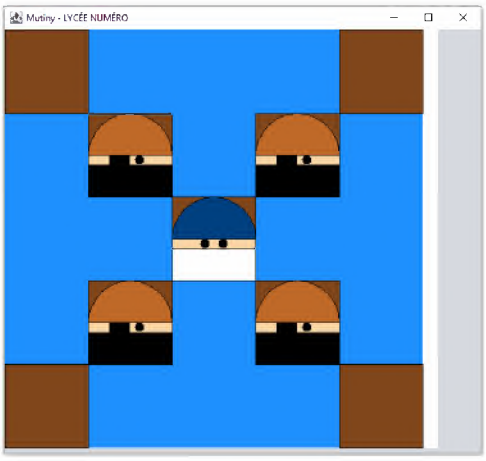
Préliminaire :

- Dans le présent projet, ne créez pas d'autres attributs que ceux indiqués dans le questionnaire et illustrés dans le diagramme UML sur la dernière page.
- Chaque classe fille doit, pour autant que possible, se servir des méthodes de sa classe mère.
- Ajoutez, tout en haut de chaque classe, votre code de l'examen en commentaire.

En cas de non-respect de la nomenclature, des conventions du cours, des indentations, des indications de l'UML, etc., jusqu'à 3 points pourront être retranchés de la note finale.

Principe du jeu « Mutiny »

Dans la suite vous allez développer le jeu « Mutiny » (*FR : mutinerie, DE : Meuterei*). Dans ce jeu, les marins sont entourés de pirates. Le but est de déplacer les figurines sur un plateau quadrillé de taille $n \times n$ en un minimum de coups, conformément aux règles ci-dessous, et de transformer tous les pirates en marins !



- Les marins et les pirates ne peuvent se déplacer qu'en diagonale sur les cellules brunes.
- Le joueur doit cliquer deux fois pour effectuer un mouvement : d'abord sur une figurine pour la sélectionner, puis sur une cellule vide de la **même** diagonale.
- Il est possible de sauter par-dessus plusieurs marins et pirates en un seul coup.
- Les pirates sautés deviennent des marins et les marins sautés deviennent des pirates.
- La partie est terminée lorsqu'il ne reste plus que des marins ou que des pirates. Cependant, la partie est gagnée uniquement s'il ne reste que des marins.

Ouvrez le projet *Mutiny* dans Netbeans.

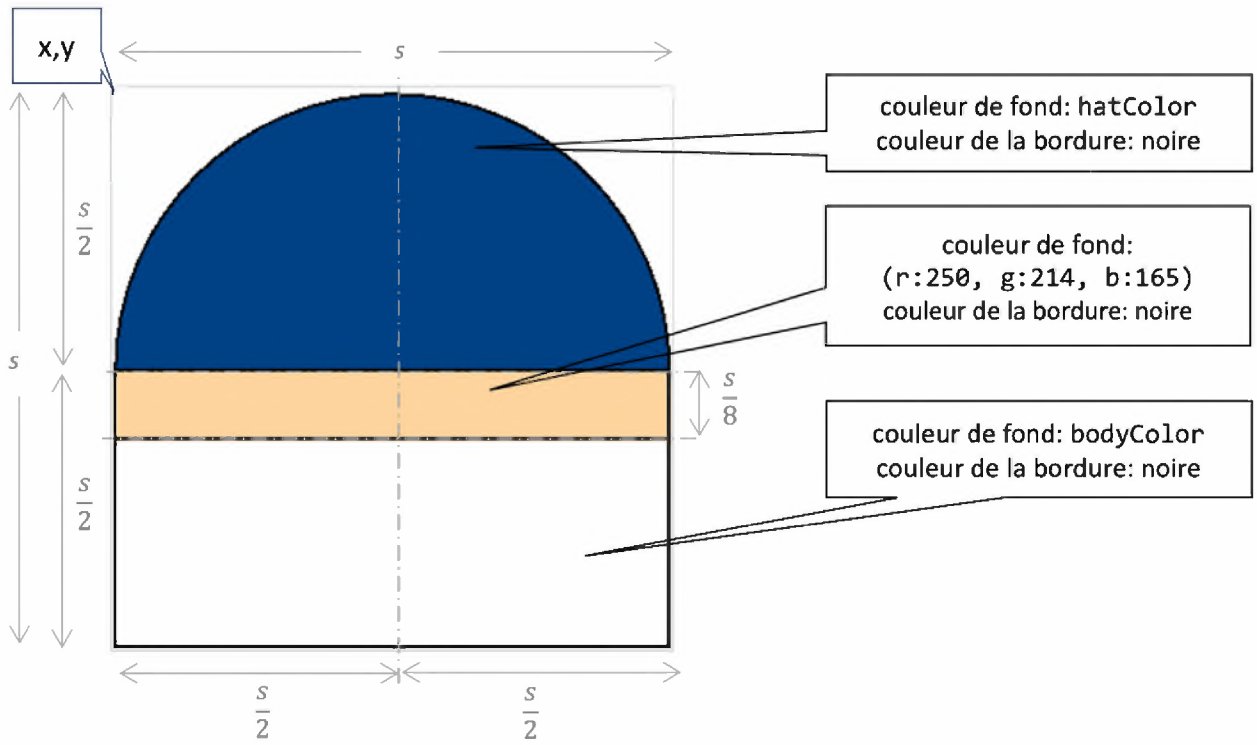
La classe *Character* *(5 points)*

Complétez la classe `Character` qui représente une figurine sur le terrain de jeu et agit comme classe mère pour un pirate et un marin. Une figurine est caractérisée par les attributs suivants :

- `row` - la ligne du terrain de jeu où la figurine est positionnée dont $row \in [0, n[$;
- `col` - la colonne du terrain de jeu où la figurine est positionnée dont $col \in [0, n[$;
- `hatColor` - la couleur du chapeau de la figurine ;
- `bodyColor` - la couleur du corps de la figurine.

Méthodes

- `jump` affecte les valeurs transmises en paramètre aux attributs `col` et `row`. **(0,5p)**
- `draw` dessine l'ensemble de la figurine (corps, chapeau, visage, à l'exception des yeux) à la bonne position sur le terrain de jeu. Le paramètre *s* représente la taille d'une cellule dans la grille. Il est ainsi possible de calculer les coordonnées x et y d'une figurine à dessiner. La figurine est dessinée comme représentée dans l'image suivante. **(4,5p)**



La classe *Sailor* *(3 points)*

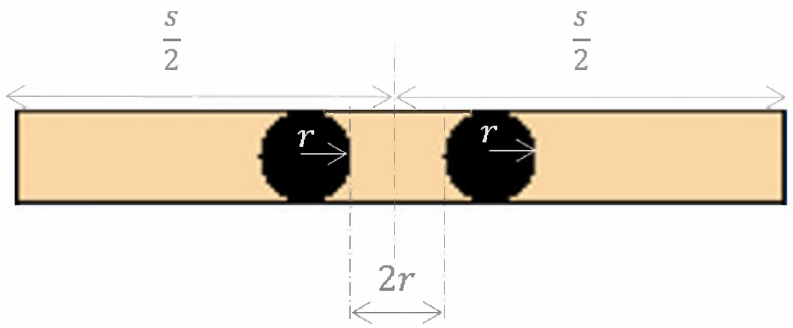
Implémentez la classe `Sailor` qui représente un marin en vous basant sur le diagramme UML donné (voir dernière page) et les indications supplémentaires ci-dessous.

Constructeur ***(1p)***

Le constructeur permet d'initialiser les attributs. Utilisez la couleur blanche pour `bodyColor` et la couleur `(r:191, g:105, b:40)` pour `hatColor`.

Méthodes ***(2p)***

La méthode `draw` rajoute deux yeux noirs d'un rayon de $r = \frac{s}{16}$ à l'objet dessiné comme illustré ci-dessous.



La classe *Pirate* *(3 points)*

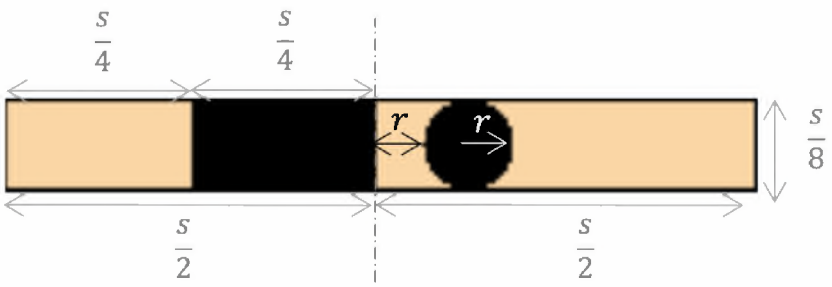
Implémentez la classe `Pirate` qui représente un pirate en vous basant sur le diagramme UML donné (voir dernière page) et les indications supplémentaires ci-dessous.

Constructeur ***(1p)***

Le constructeur permet d'initialiser les attributs. Utilisez la couleur noire pour `bodyColor` et la couleur `(r:191, g:105, b:40)` pour `hatColor`.

Méthodes ***(2p)***

La méthode `draw` rajoute un cache-œil et un œil noir à l'objet dessiné comme illustré ci-dessous. Le rayon de l'œil : $r = \frac{s}{16}$



La classe *GameManager* *(44 points)*

Complétez la classe `GameManager` en vous basant sur les indications ci-dessous.

Attributs

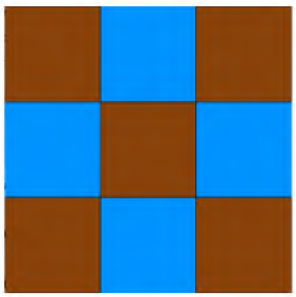
- `alCharacters` : la liste des figurines.
- `n` : le nombre de lignes et de colonnes de la grille. La numérotation débute à partir de 0.
- `side` : la taille en pixels d'une cellule du terrain de jeu quadrillé.
- `selectedCharacter` : la figurine actuellement sélectionnée, qui doit être déplacée.

Méthodes

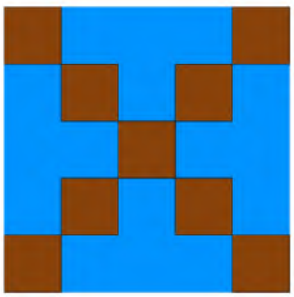
- `isPrime` indique si le nombre entier passé en paramètre est un nombre premier ou non. **(4p)**
- `getRandomPrimeNumber` prend deux nombres entiers en paramètre et retourne un nombre premier aléatoire entier compris entre ces deux nombres. Vous pouvez supposer qu'il y a toujours au moins un nombre premier dans l'intervalle donné. **(3p)**
- Le constructeur permet d'initialiser l'attribut n avec un nombre premier aléatoire dans l'intervalle [5,15]. La valeur de l'attribut `side` dépend de la taille de la surface de dessin. La grille entière - située en haut à gauche du canevas - est toujours de forme carrée maximale, même si la surface de dessin n'est pas carrée. L'attribut `selectedCharacter` est initialisé à null. Un marin est à placer au milieu du terrain, entouré de 4 pirates, comme indiqué dans la capture d'écran de la première page. **(3p)**
- `find` retourne la figurine située aux coordonnées spécifiées par les paramètres et `null` si aucune figurine ne se trouve à ces coordonnées. **(6p)**
- `isFree` retourne `true` s'il n'y a aucune figurine aux coordonnées spécifiées par les paramètres. **(1p)**
- `isWithinBoard` retourne `true` si les coordonnées spécifiées par les paramètres sont à l'intérieur de la grille. **(1p)**
- `isInDiagonalFields` retourne `true` si les coordonnées spécifiées par les paramètres se trouvent sur l'une ou l'autre des deux diagonales de la grille carrée. **(1p)**
- `isDiagonalJump` reçoit les coordonnées de départ et d'arrivée en paramètre et détermine si le déplacement peut se faire dans une direction diagonale. Si c'est le cas, elle retourne `true`, sinon elle retourne `false`. **(1,5p)**
- `isValidJump` retourne `true` si un saut est possible (voir principe du jeu en page 1). **(2p)**
- `drawBoard` dessine l'ensemble du terrain (voir exemples page suivante) : **(3p)**
  - tout le terrain du jeu est dessiné en bleu `(r:30, g:144, b:255)` ;
  - les cellules situées sur les axes diagonaux sont affichées en brun avec une bordure noire. Utilisez la couleur `(r:128, g:70, b:27)`.

Exemples :

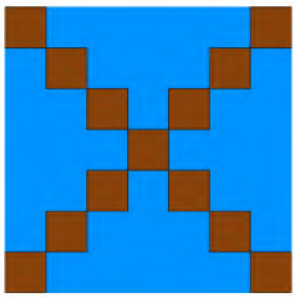
n = 3



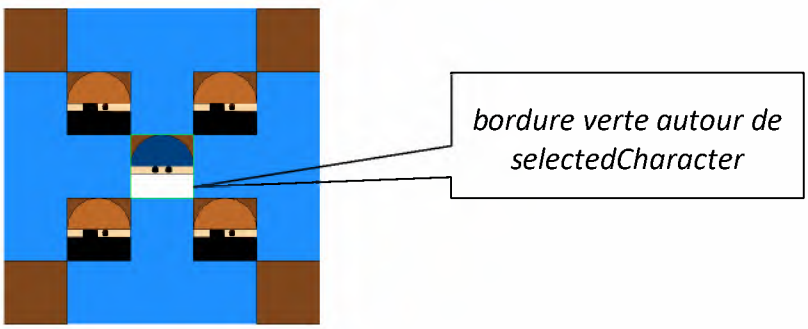
n = 5



n = 7



- `addCharacter` ajoute la figurine donnée à la liste des figurines, seulement dans le cas où les coordonnées se trouvent sur l'une des deux diagonales de la grille et si la cellule indiquée par les coordonnées est libre. **(2p)**
- `switchTypes` reçoit les coordonnées de départ et d'arrivée en paramètre et échange le type de toutes les figurines entre la cellule de départ et la cellule cible sur la diagonale. Les marins seront remplacés par des pirates et vice versa. **(6p)**
- `selectCharacter` reçoit un point du terrain de jeu. **(4p)**
  - Si l'utilisateur n'a pas sélectionné de figurine pour l'instant (utilisez l'attribut `selectedCharacter` pour contrôler cela), alors l'attribut `selectedCharacter` contiendra la figurine de la cellule cliquée.
  - Si l'utilisateur a déjà sélectionné une figurine, cela signifie que l'utilisateur souhaite déplacer la figurine sélectionnée vers une cellule cible.
    - La méthode déplace alors la figurine sélectionnée vers la cellule cible uniquement s'il s'agit d'un mouvement valide.
    - Avant d'effectuer le mouvement, le type des figurines entre la cellule de départ et la cellule cible sont échangés.
    - Dans tous les cas, `selectedCharacter` est remis à `null`.
- `drawAll` dessine la terrain du jeu, les cases diagonales et toutes les figurines. Si une figurine est sélectionnée, la cellule concernée est dessinée avec une bordure verte. **(3p)**



- `getNumberOfPirates` retourne le nombre de pirates sur le terrain de jeu. **(2p)**
- `isGameOver` retourne `true` si la partie est terminée et `false` sinon. Le jeu est considéré comme terminé, s'il ne reste qu'un seul type de figurines sur le terrain de jeu. **(1p)**
- `isGameWon` retourne `true` si le jeu est gagné et `false` sinon. Le jeu est considéré comme gagné, s'il ne reste plus que des marins. **(0,5p)**

La classe *DrawPanel* *(2 points)*

Complétez la classe `DrawPanel` en ajoutant son attribut et manipulateur requis. Ajoutez la méthode `paintComponent` qui dessine, si possible, le jeu sur un fond blanc.

La classe *MainFrame* *(3 points)*

L'interface du jeu est donnée et la classe doit être complétée. La classe dispose de l'attribut gameManager qui représente le jeu. Ajoutez à la fiche principale le titre du programme « *Mutiny* - » suivi de votre numéro de candidat.

Constructeur ***(1p)***

Le constructeur initialise le jeu aux dimensions du canevas.

Méthodes ***(2p)***

Ajoutez un événement de type « `mouseReleased` » au canevas. Chaque fois que l'utilisateur relâche un bouton de la souris et que le jeu n'est pas encore terminé, la méthode `selectCharacter` de `gameManager` est appelée et le jeu est mis à jour. Si le jeu est maintenant terminé, le titre du jeu doit être remplacé par l'une des informations suivantes :

- « *Game Over ! You won !* », si le jeu est gagné ;
- « *Game Over ! You lost !* », si le jeu est perdu.

Annexes

Enseignement secondaire général
Division technique générale – Section technique générale
Examen 1GIG

Liste des composants et classes connus

Liste des composants (propriétés, événements et méthodes) et classes à connaître pour l'épreuve en informatique à l'examen de fin d'études secondaires générales - division technique générale.

Package	Classe	Details	Remarques / Constantes
javax.swing	JFrame	Méthodes - setTitle(...) / getTitle() NetBeans Object Inspector Property - title	
	JButton JLabel JTextField	Méthodes - setText(...) / getText() - setVisible(...) - setEnabled(...) Événement - actionPerformed NetBeans Object Inspector Property - icon	- le libellé *JLabel* peut aussi être utilisé pour visualiser des images via la propriété « icon » de l'inspecteur objet de NetBeans (le composant *JTextField* ne possède pas de propriété « icon »).
	JSlider	Méthodes - setMinimum(...) / getMinimum() - setMaximum(...) / getMaximum() - setValue(...) / getValue() Événement - stateChanged	
	JPanel	Méthodes - setVisible(...) - setBackground(...) / getBackground() - getWidth() / getHeight() - paintComponent(Graphics g) - repaint() Événements - MousePressed / MouseReleased - MouseDragged / MouseMoved	- *JPanel* est utilisé pour regrouper d'autres composants visuels et pour réaliser des dessins. - Lors de la réalisation de dessins, la méthode **`public void paintComponent (Graphics g)`** est à surcharger.
javax.swing	JList	Méthodes - setListData(...) - getSelectedIndex() / setSelectedIndex(...) Événement - valueChanged NetBeans Object Inspector - *Properties* - model - selectionMode (SINGLE) NetBeans Object Inspector - *Code* - Type Parameters : - *(vide)*	- *JList* est utilisé surtout pour afficher le contenu d'une liste *ArrayList*.
java.awt.event	ActionEvent	- Ce type d'objet est uniquement utilisé dans les méthodes de réaction ajoutées de manière automatique à l'aide de NetBeans.	
	MouseEvent	Méthodes - getX() / getY() - getPoint() - getButton()	Constantes - BUTTON1 - BUTTON2 - BUTTON3

Package	Classe	Details	Remarques
javax.swing	Timer	Constructeur - Timer(int,ActionListener) Méthodes - start() - stop() - setDelay(...) - isRunning()	
		- Comme *ActionListener* on utilisera de préférence celui d'un bouton. Exemple : `timer = new Timer(1000,stepButton.getActionListeners()[0]);`	
java.awt	Graphics	Méthodes - drawLine(...) - drawOval(...) / fillOval(...) - drawRect(...) / fillRect(...) - drawString(...) - setColor(...) / getColor()	
	Color	Constructeurs - Color(...)	
	Point	Constructeurs - Point(...) Attributs (publics) - x et y Méthodes - getLocation() / setLocation(...)	
java.util	ArrayList	Méthodes - add(...) - clear() - contains(...) - get(...) - indexOf(...) - remove(...) - set(...) - size() - isEmpty() - toArray()	- `Object[] toArray()` est employé uniquement pour passer les contenus d'une liste à la méthode *setListData(...)* d'une *JList*.
java.lang	String	Méthodes - equals(...) / compareTo(...) - contains(...) - valueOf(...)	
	Integer Double	Méthodes - equals(...) / compareTo(...) - valueOf(...)	
	Math	Méthodes - abs(...) - round(...) - random() - sqrt(...) - pow(...) - sin(...), cos(...), tan(...)	Constante: - PI
	System	Méthode - out.print() - out.println()	

Diagramme UML