LE GOUVERNEMENT
DU GRAND-DUCHÉ DE LUXEMBOURG
Ministère de l'Éducation nationale,
de l'Enfance et de la Jeunesse

EXAMEN DE FIN D'ÉTUDES SECONDAIRES TECHNIQUES
2017

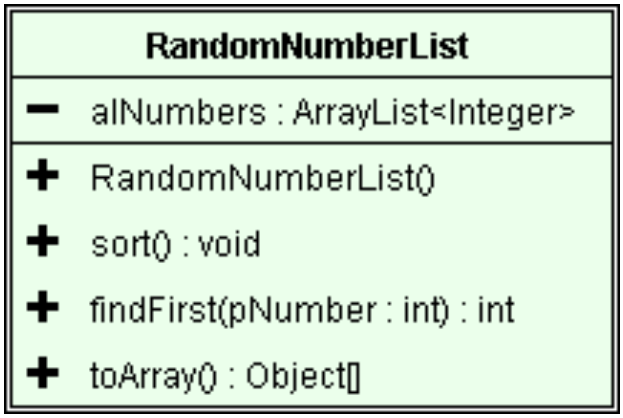
BRANCHE	SECTION(S)	ÉPREUVE ÉCRITE
Informatique	**GE**	*Durée de l'épreuve* 3h
		Date de l'épreuve 12/06/2017
		Numéro du candidat

Dans votre répertoire de travail (à définir par chaque Lycée), vous trouverez un dossier nommé **EXAMEN_GE**. Renommez ce dossier en remplaçant le nom par votre code de l'examen (Exemple de notation : **LTXY_GE1_07**). *Tous vos fichiers devront être sauvegardés à l'intérieur de ce dossier, qui sera appelé 'votre dossier' par la suite !*

Question 1 — 12 p.

Ouvrez le projet **Question1** de votre dossier. L'environnement de programmation est à votre choix (**Unimozer** ou **NetBeans**).

La classe **RandomNumberList** possède une liste **alNumbers** qui sera remplie avec des nombres aléatoires.

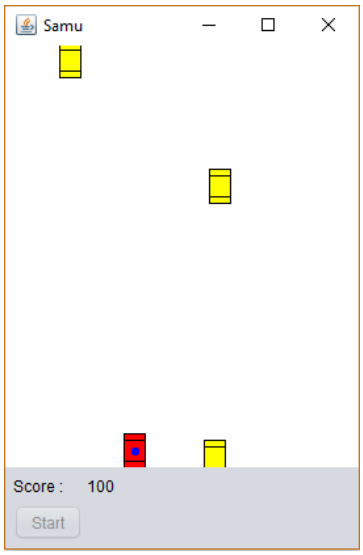


Travail à faire :

- Programmez la méthode **sort()** qui permet de faire un tri de la liste en ordre croissant en utilisant l'algorithme de tri par sélection directe. 8 p.
- Programmez la méthode **findFirst(…)** qui retourne la **position** du premier élément de la liste qui correspond à la valeur du paramètre **pNumber** en utilisant la recherche séquentielle. Si la valeur de **pNumber** ne se trouve pas dans la liste, la méthode doit retourner la valeur **-1**. 4 p.

Question 2 — 5+10+6+12+3+12 = 48 pts

Dans la suite vous allez développer l'application « **Samu** ». Il s'agit d'un petit programme qui simule une voiture d'intervention (Samu) qui doit essayer d'éviter la collision avec des voitures qui roulent à contre-sens.



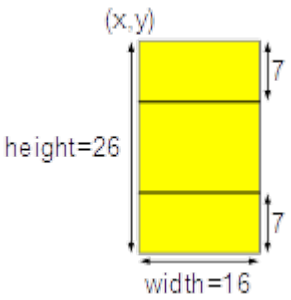
Vous trouvez une version exécutable du programme, nommé **Modele.jar**, dans le dossier « Question2 » de votre dossier. Avant de continuer, il est recommandé de lancer et de tester ce programme. Le samu peut être déplacé vers la gauche en gardant le bouton gauche de la souris enfoncé et vers la droite en gardant le bouton droit enfoncé sur la surface de dessin. Dès que le bouton de la souris est relâché, le samu ne se déplace plus.

Créez dans **NetBeans** un nouveau projet nommé **Samu** que vous sauvegardez dans le dossier **Question2** de **votre dossier**.

Réalisez cette application **en vous basant sur la version exécutable fournie** ainsi que sur le **diagramme UML** de la page 5 et tout en **respectant les instructions et précisions données dans la suite**.

La classe Vehicle (5 pts)

- Cette classe est la **classe mère** des classes **Samu** et **Car**.
- Les attributs **x** et **y** représentent les coordonnées du **coin supérieur gauche** d'un véhicule.
- Les attributs **width** et **height** représentent les dimensions du véhicule et sont initialisés à 16 (width) et 26 (height).
- L'attribut **color** sert à mémoriser la couleur du véhicule.
- Le **constructeur** initialise les coordonnées du véhicule avec les valeurs passées comme paramètre.
- Ajoutez les accesseurs (getters) et manipulateurs (setters) spécifiés dans le schéma UML.
- La méthode **draw(…)** dessine le véhicule avec la couleur de l'attribut **color** selon le schéma ci-dessous. Tous les bords sont dessinés en noir.



La classe Samu (6 pts)

- Cette classe hérite de la classe **Vehicle** et sert à représenter le **samu**.
- L'attribut **stepX** correspond au pas de déplacement horizontal du **samu**. Il est initialisé à 0.
- Le **constructeur** initialise les coordonnées du **samu** avec les valeurs reçues comme paramètres et initialise la couleur avec du rouge.
- La méthode **setStep(…)** sert à modifier la valeur de **stepX**.
- La méthode **move(…)** prend comme paramètre la largeur du canevas et sert à faire bouger le **samu**. Veillez à ce que le **samu** ne puisse pas dépasser le bord gauche ou droit.
- La méthode **draw(...)** utilise la méthode **draw(…)** de la classe mère **Vehicle** pour dessiner le **samu**. En plus cette méthode dessine un disque bleu d'un diamètre de 7 points au milieu du **samu** pour simuler le gyrophare.

La classe Car (10 pts)

- Cette classe hérite de la classe **Vehicle** et sert à représenter les voitures jaunes qui descendent sur la surface de dessin.
- L'attribut **stepY** définit le pas de déplacement vertical de la voiture.
- Le **constructeur** initialise les coordonnées de la voiture avec les valeurs reçues comme paramètres, initialise la couleur à jaune et initialise **stepY** à une valeur aléatoire entre 2 et 5 (bornes incluses).
- La méthode **isCollision(…)** détecte si une collision avec le **samu** passé comme paramètre a eu lieu. Une collision s'est produite si la voiture et le **samu** se superposent.
- La méthode **move(…)** prend en paramètres les dimensions de la surface de dessin et un objet de type **Samu**. Cette méthode fait bouger la voiture de la distance **stepY** vers le bas. Selon la situation, la méthode retourne une des valeurs suivantes :
  - 0 : rien de particulier.
  - 1 : la voiture a complètement quitté la surface de dessin.
  - 2 : une collision avec le **samu** est intervenue. La détection de cette collision est réalisée à l'aide de la méthode **isCollision(…)** décrite ci-dessus.

La classe Game (12 pts)

- La classe **Game** est responsable de la gestion du jeu. Cette classe est représentée par
  - une liste **alCars** contenant les voitures jaunes ;
  - un attribut **samu** qui représente le samu ;
  - un attribut **score** qui représente les points obtenus par le joueur (initialisé à zéro).
- Le constructeur crée un nouveau **samu** au centre du bord inférieur de la surface de dessin. Les dimensions de la surface de dessin sont passées comme paramètres au constructeur.
- La méthode **size()** retourne la taille de la liste.
- L'accesseur **getScore()** retourne le score du joueur.
- La méthode **setSamuStepX(…)** permet de modifier le pas de déplacement du **samu**.
- La méthode **createCar(…)** crée une nouvelle voiture et l'ajoute à la liste **alCars**. Sa position horizontale est aléatoire mais doit se trouver complètement à l'intérieur de la surface de dessin. Sa position verticale est complètement au-dessus de la surface de dessin, donc à -26 (hauteur de la voiture).
- La méthode **reset(…)** remet le jeu à zéro. Le score est remis à 0, la liste des voitures est vidée et un nouveau samu est créé.

- La méthode **move(…)** fait bouger le **samu** ainsi que toutes les voitures. En ce qui concerne le mouvement des voitures, la valeur de retour de la méthode **move(…)** de la classe **Car** est utilisée pour modifier l'état du jeu :
  - une voiture qui sort du jeu (code = 1) est retirée de la liste et le score est incrémenté de 100 points ;
  - si une collision avec le **samu** est détectée, la méthode retourne la valeur **true**.
- La méthode **draw(…)** dessine toutes les voitures ainsi que le **samu**.

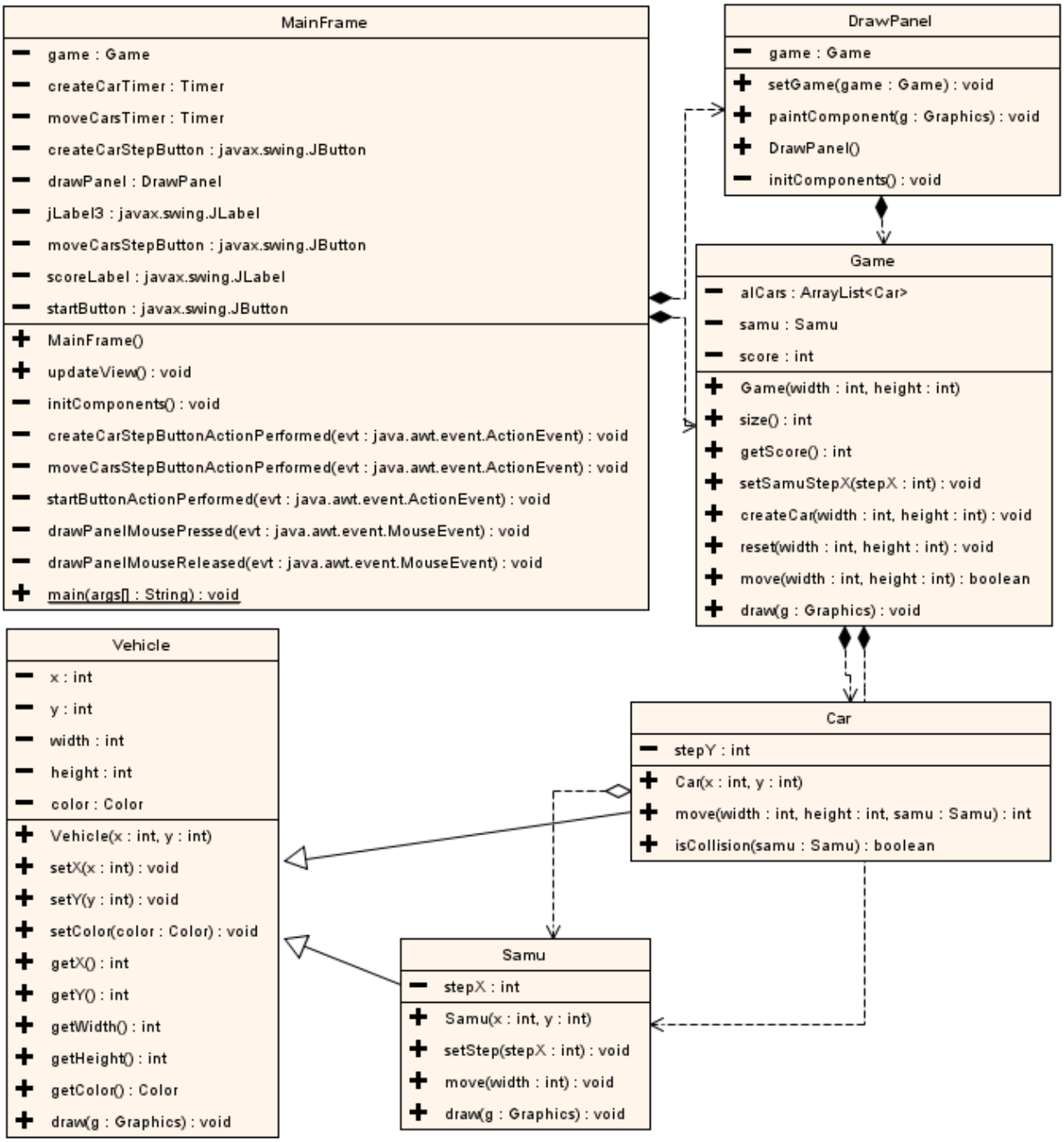
La classe DrawPanel (3 pts)

- Cette classe possède un attribut **game** et un manipulateur **setGame(…)**.
- La méthode **paintComponent(…)** dessine d'abord l'arrière-plan blanc, puis, si possible, les différents véhicules du jeu par un appel à la méthode **draw(…)** de l'attribut **game**.

La classe MainFrame (12 pts)

- Réalisez l'interface graphique selon le modèle et le schéma UML.
- La classe **MainFrame** possède un attribut **game** ainsi que deux chronomètres.
- Complétez le constructeur **MainFrame()** par les instructions nécessaires pour initialiser l'attribut **game** ainsi que les deux chronomètres et informer le **drawPanel** de l'instance de **game** :
  - le chronomètre **createCarTimer** est responsable de la création de nouvelles voitures et actionne le bouton **createCarStepButton** toutes les secondes ;
  - le chronomètre **moveCarsTimer** est responsable du mouvement des voitures et du **samu** et actionne le bouton **moveCarsStepButton** toutes les 10 millisecondes ;
  - les boutons **createCarStepButton** et **moveCarStepButton** sont rendus invisibles ;
  - le titre de l'application est « Samu ».
- La méthode **updateView()** rafraîchit la surface de dessin et met à jour l'affichage du score.
- Lorsqu'on clique sur le bouton **Start**, celui-ci est désactivé, le jeu est réinitialisé et les deux chronos sont activés.
- Le bouton **createCarStepButton(…)** crée une nouvelle voiture en indiquant les dimensions de la surface de dessin.
- Le bouton **moveCarsStepButton(…)** fait bouger les véhicules. Si une collision a été détectée, les chronomètres sont arrêtés et le bouton **Start** est réactivé.
- Le fait de cliquer sur la surface de dessin fait bouger le **samu**. Selon le bouton de la souris qui est pressé, la valeur **-1** (mouvement vers la gauche) ou **1** (mouvement vers la droite) est transmis à **setSamuStep(…)**. Le fait de relâcher le bouton de la souris fait transmettre **0** à **setSamuStep(…)**.

Schéma UML


MainFrame
- game : Game
- createCarTimer : Timer
- moveCarsTimer : Timer
- createCarStepButton : javax.swing.JButton
- drawPanel : DrawPanel
- jLabel3 : javax.swing.JLabel
- moveCarsStepButton : javax.swing.JButton
- scoreLabel : javax.swing.JLabel
- startButton : javax.swing.JButton
+ MainFrame()
+ updateView() : void
- initComponents() : void
- createCarStepButtonActionPerformed(evt : java.awt.event.ActionEvent) : void
- moveCarsStepButtonActionPerformed(evt : java.awt.event.ActionEvent) : void
- startButtonActionPerformed(evt : java.awt.event.ActionEvent) : void
- drawPanelMousePressed(evt : java.awt.event.MouseEvent) : void
- drawPanelMouseReleased(evt : java.awt.event.MouseEvent) : void
+ main(args[] : String) : void
DrawPanel
- game : Game
+ setGame(game : Game) : void
+ paintComponent(g : Graphics) : void
+ DrawPanel()
- initComponents() : void
Game
- alCars : ArrayList<Car>
- samu : Samu
- score : int
+ Game(width : int, height : int)
+ size() : int
+ getScore() : int
+ setSamuStepX(stepX : int) : void
+ createCar(width : int, height : int) : void
+ reset(width : int, height : int) : void
+ move(width : int, height : int) : boolean
+ draw(g : Graphics) : void
Vehicle
- x : int
- y : int
- width : int
- height : int
- color : Color
+ Vehicle(x : int, y : int)
+ setX(x : int) : void
+ setY(y : int) : void
+ setColor(color : Color) : void
+ getX() : int
+ getY() : int
+ getWidth() : int
+ getHeight() : int
+ getColor() : Color
+ draw(g : Graphics) : void
Car
- stepY : int
+ Car(x : int, y : int)
+ move(width : int, height : int, samu : Samu) : int
+ isCollision(samu : Samu) : boolean
Samu
- stepX : int
+ Samu(x : int, y : int)
+ setStep(stepX : int) : void
+ move(width : int) : void
+ draw(g : Graphics) : void

Enseignement secondaire technique
Division technique générale
Examen 13GE

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 techniques - 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* il faut utiliser celui d'un bouton. Exemple : `Timer 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()	