Code branche **INFOR**	Ministère de l'Éducation nationale, de l'Enfance et de la Jeunesse **EXAMEN DE FIN D'ÉTUDES SECONDAIRES TECHNIQUES** Régime technique – Session 2015	
Épreuve écrite	Branche	Division / Section
Durée de l'épreuve **3h**	**Informatique**	**GE**
Date de l'épreuve Repêchage 5 juin 2015		

Dans votre répertoire de travail (à définir par chaque lycée), vous trouverez un sous-dossier nommé **EXAMEN_GE**. Renommez ce dossier en remplaçant le nom par votre code de l'examen (Exemple de notation : **LTXY_GE2_07**). *Tous vos fichiers devront être sauvegardés à l'intérieur de ce sous-dossier, qui sera appelé 'votre dossier' dans la suite !*

Question 1 : LOTTO — *4+8 = 12 points*

Dans NetBeans ouvrez le projet **Question1** qui contient la classe **Lotto** et qui simule des tirages de lotto.

Un clic sur le bouton **'generate sequence'** génère et affiche une nouvelle séquence de 6 nombres aléatoires compris entre 1 et 49. Dans cet exercice on admettra que des doublons peuvent se produire.

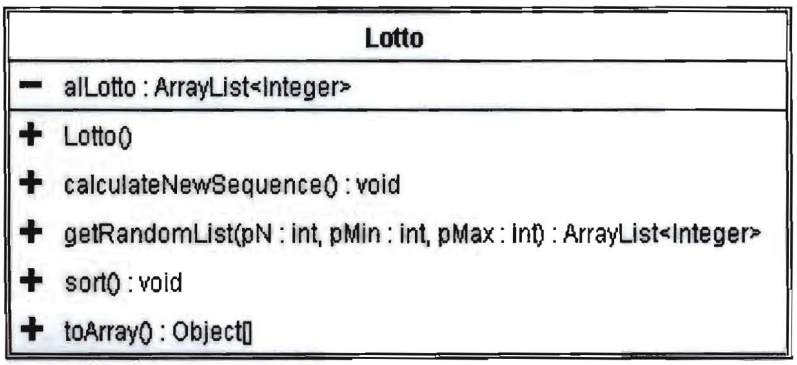
Un clic sur le bouton **'sort sequence'** trie la séquence (liste) des nombres.

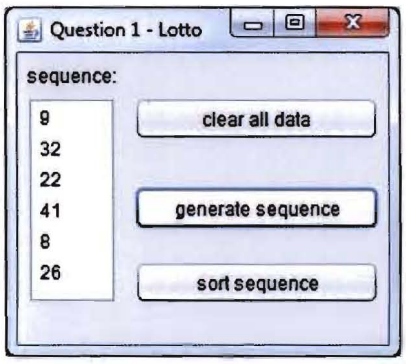
L'exécutable **Question1.jar** vous illustre le fonctionnement.

Dans le fichier **Lotto.java** réalisez la méthode **getRandomList()** qui calcule et retourne une liste de **pN** nombres entiers aléatoires compris entre **pMin** et **pMax** (bornes incluses).

Dans le fichier **Lotto.java** réalisez la méthode **sort()** qui trie la liste **alLotto** à l'aide de l'algorithme de tri croissant par sélection directe.

N. B. : Les actions relatives aux boutons sont déjà programmées, de façon que vous puissiez immédiatement vérifier le bon fonctionnement de vos méthodes.





Question 2 : Figures — *18+10+4+16 = 48 points*

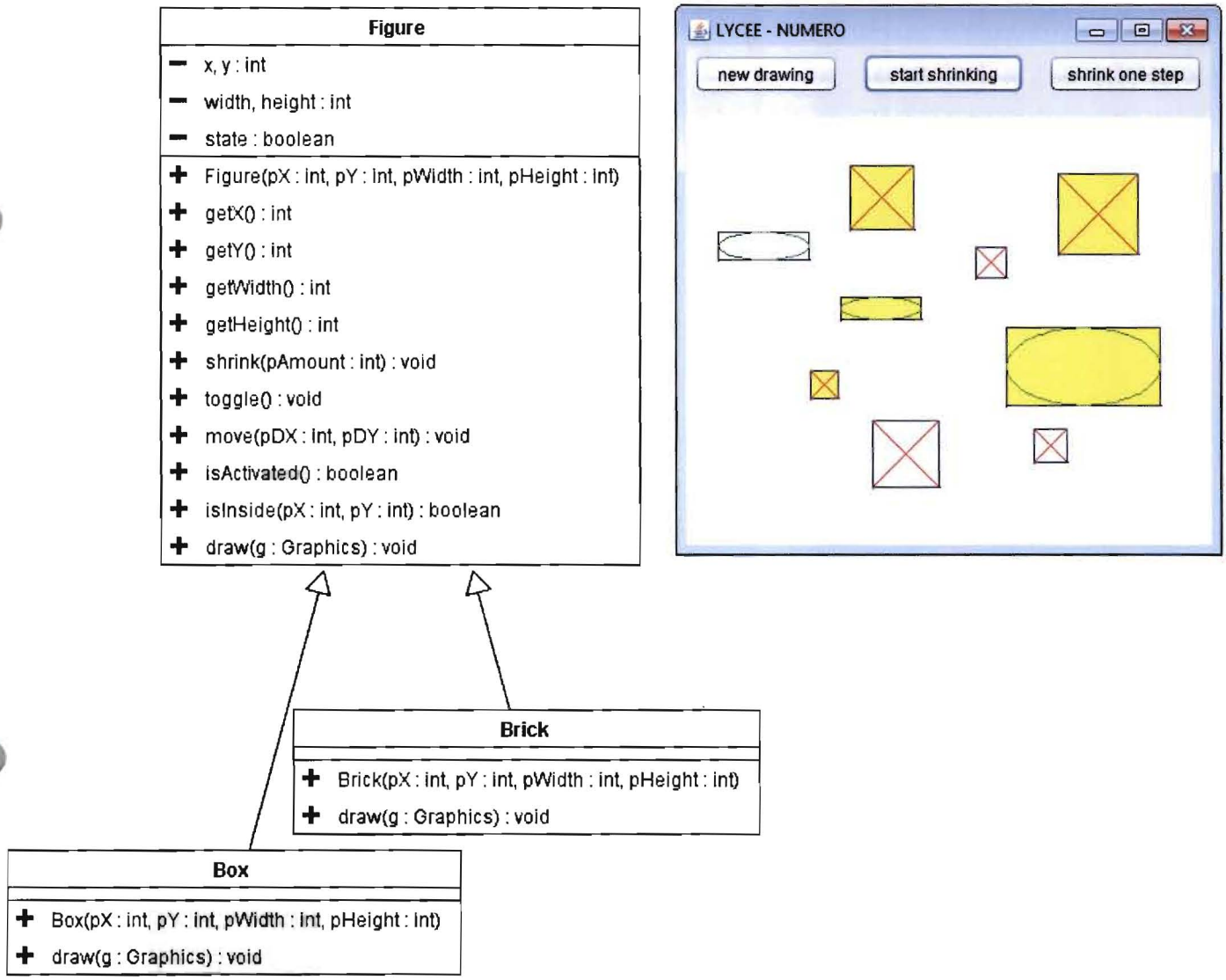
Fermez le projet Question1 et créez **dans votre dossier** le nouveau projet **Question2**.

Ce projet permet de gérer des figures géométriques simples.

Servez-vous de l'exécutable Question2.jar pour vous familiariser avec les détails de son fonctionnement.

Partie 1 – Figure.java, Box.java, Brick.java, (12+3+3 = 18 points)

Ajoutez à votre projet la classe générique **Figure** et les classes dérivées **Box** et **Brick** comme décrites dans le diagramme UML ci-dessous.



La classe **Figure** dispose des attributs privés suivants :

x, y	les coordonnées du centre de la figure
width, height	la largeur et la hauteur de la figure
state	l'état de la figure (activée ou non)

En outre des accesseurs standard, la classe **Figure** dispose des méthodes publiques suivantes :

constructeur	Il initialise les attributs par les valeurs des paramètres respectifs. Au début l'état sera désactivé (false).
shrink(int pAmount)	Elle fait réduire chacune des largeur et hauteur de pAmount.
toggle()	Elle change l'état de la figure.
move(int pDX, int pDY)	Elle fait déplacer la figure selon les distances fournies par paramètres.
isActivated()	C'est l'accesseur standard de l'état de la figure.
isInside(int pX, int pY)	Elle vérifie si oui ou non les coordonnées fournies comme paramètres se trouvent à l'intérieur (au sens large, donc bords inclus) de la figure.
draw(Graphics g)	Elle dessine la figure sur le canevas g sous forme d'un rectangle bleu. Si la figure est activée, son intérieur sera jaune, sinon il sera creux.

La classe **Box**, dérivée de la classe **Figure** dispose des méthodes publiques suivantes :

constructeur	Il initialise les attributs par les valeurs des paramètres respectifs et l'état est désactivé. A cet effet il fera appel au constructeur de sa classe-mère.
draw(Graphics g)	Elle dessine la figure sure le canevas g. En plus du rectangle bleu (comme décrit plus haut), les deux diagonales sont ajoutées en rouge.

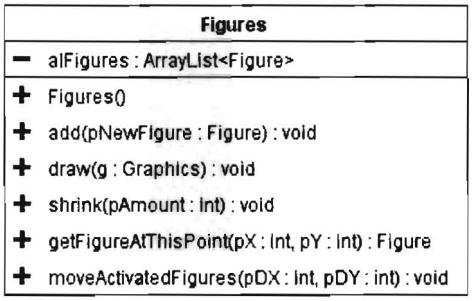
La classe **Brick**, dérivée de la classe **Figure** dispose des méthodes publiques suivantes :

constructeur	Il initialise les attributs par les valeurs des paramètres respectifs et l'état est désactivé. A cet effet il fera appel au constructeur de sa classe-mère.
draw(Graphics g)	Elle dessine la figure sur le canevas g. En plus du rectangle bleu (comme décrit plus haut), une ellipse verte, creuse, inscrite au rectangle est ajoutée.

Partie 2 – Figures.java (10 points)

Ajoutez à votre projet la classe **Figures** comme décrit dans le diagramme UML ci-contre.

Elle dispose uniquement de l'attribut privé **alFigures** qui contient les différentes figures à gérer.



La classe **Figures** dispose des méthodes publiques suivantes :

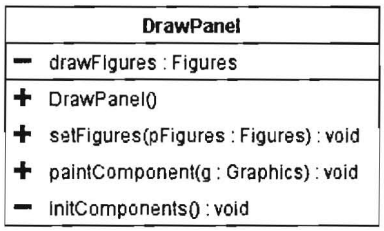
constructeur	Il initialise la liste **alFigures**.
add(Figure pNewFigure)	Elle ajoute la figure passée comme paramètre à la liste alFigures.
draw(Graphics g)	Elle dessine toutes les figures de la liste sur le canevas g.
shrink(int pAmount)	Elle réduit toutes les figures de la liste par valeur pAmount, passée comme paramètre. Si, après une telle réduction, la largeur ou la hauteur d'une figure tombe en-dessous de 2 pixels, alors cette figure-là est enlevée de la liste.
getFigureAtThisPoint(int pX, int pY)	Elle vérifie si les coordonnées indiquées se trouvent à l'intérieur d'une figure. Si oui, la première figure trouvée est retournée, sinon la valeur null est retournée.
moveActivatedFigures(int pDX, int pDY)	Elle fait déplacer toutes les figures activées selon les distances fournies par paramètres.

Partie 3 – DrawPanel.java — 4 points

Ajoutez à votre projet la classe **DrawPanel** comme décrit dans le diagramme UML ci-contre. Elle accueillera le dessin des figures.

La classe **DrawPanel** dispose uniquement de l'attribut privé **drawFigures**.

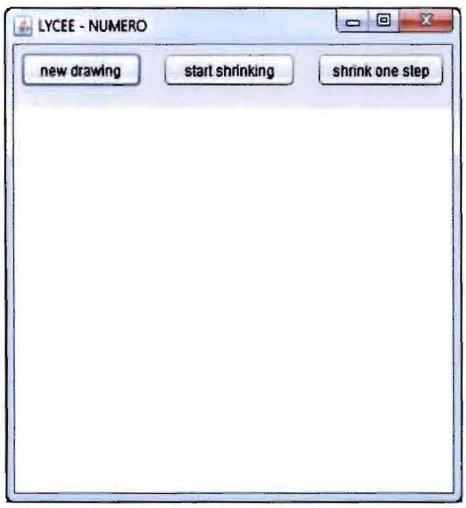
La classe **DrawPanel** dispose des méthodes publiques suivantes :



constructeur	Il est défini d'office et ne nécessite pas de modifications.
setFigures(Figures pFigures)	Elle sert à faire le lien entre l'attribut privé **drawFigures** et l'attribut privé **mainFigures** de la fiche principale de l'application.
paintComponent	Elle dessine sur fond blanc toutes les figures, pour autant que la classe Figures soit initialisée (instanciée).

Partie 4 – MainFrame.java — 16 points

Ajoutez à votre projet la classe **MainFrame** comme décrit dans le diagramme UML (voir page suivante) et réalisez l'interface en vous tenant fidèlement à la copie d'écran illustrée ci-contre (2pts).



Inscrivez votre numéro d'examen dans le titre de la fiche.

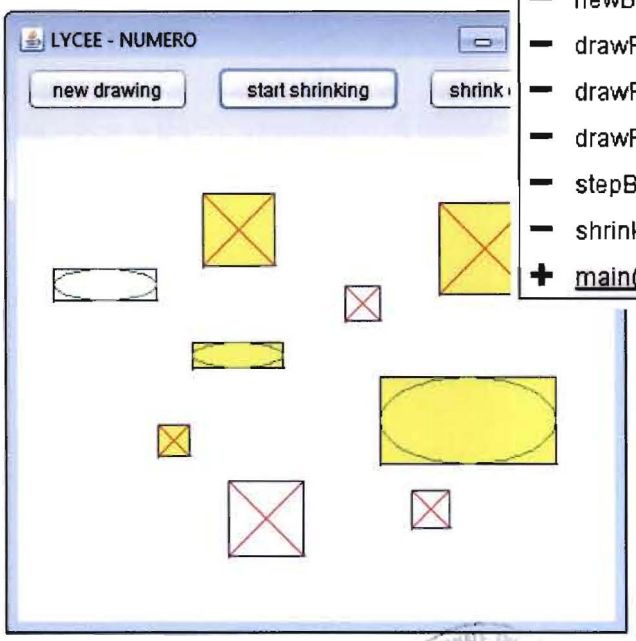
La classe **MainFrame** dispose des attributs privés suivants :

mainFigures	Il contient les figures à gérer.
oldMouseX, oldMouseY	Les anciennes coordonnées de la souris.
mainTimer	Si activé, il fait réduire (de 1 px), 10 fois par seconde, toutes les figures du dessin. Il est associé au bouton **stepButton**.

Complétez classe **MainFrame** selon les indications suivantes et gardez le graphique toujours à jour :

- Un clic sur le bouton **newButton** (**'new drawing'**) remet la fiche entière en état vierge, dessin vide et timer a l'arrêt.
- Lorsqu'on relâche le bouton de la souris sur le graphique et qu'il n'y ait pas encore de figure à cet endroit, alors une nouvelle figure aléatoire (**Box** ou **Brick**) y est créée et affichée. Sa hauteur sera aléatoire aussi et comprise entre 20 et 30 pixels. Une figure de type **Box** sera carrée, alors que la largeur d'une figure de type **Brick** sera le double de sa hauteur.
  Si par contre il y a déjà une figure à cet endroit, alors cette figure-là changera d'état.
- Lorsque la souris est déplacée avec le bouton enfoncé, alors toutes les figures activées seront déplacées selon le mouvement de la souris.
- Un clic sur le bouton **stepButton (‘shrink one step’)** fait réduire les dimensions de toutes les figures du dessin d'un pixel.
- Un clic sur le bouton **shrinkButton (‘start shrinking’)** fait démarrer le timer **mainTimer** et fait inscrire **‘stop shrinking’** sur le bouton **shrinkButton**, respectivement fait arrêter le timer **mainTimer** et fait inscrire **‘start shrinking’** sur le bouton **shrinkButton.** Lorsque le timer est actif, il fait réduire continuellement (10 fois par seconde) la taille de toutes les figures (à raison de 1 pixel par réduction).

MainFrame
− mainFigures : Figures
− mainTimer : Timer
− oldMouseX, oldMouseY : int
− drawPanel : DrawPanel
− newButton : javax.swing.JButton
− shrinkButton : javax.swing.JButton
− stepButton : javax.swing.JButton
+ MainFrame()
− initComponents() : void
− newButtonActionPerformed(evt : java.awt.event.ActionEvent) : void
− drawPanelMousePressed(evt : java.awt.event.MouseEvent) : void
− drawPanelMouseDragged(evt : java.awt.event.MouseEvent) : void
− drawPanelMouseReleased(evt : java.awt.event.MouseEvent) : void
− stepButtonActionPerformed(evt : java.awt.event.ActionEvent) : void
− shrinkButtonActionPerformed(evt : java.awt.event.ActionEvent) : void
+ main(args[] : String) : void



(page vide)

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() - setLocation(...) / getLocation() NetBeans Object Inspector Property - title	
	JButton JLabel JTextField	Méthodes - setText(...) / getText() - setLocation(...) / getLocation() - getX() / getY() - getWidth() / getHeight() - 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(...) - setEnabled(...) - setBackground(...) / getBackground() - getWidth() / getHeight() - getGraphics() - paintComponent(Graphics g) - repaint() Événements - MousePressed / MouseReleased - MouseDragged	- *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	- *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() - getLocation() - 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 tm = new Timer(1000,stepButton.getActionListeners()[0]);`	
java.awt	Graphics	Méthodes - drawLine(...) - drawOval(...) / fillOval(...) - drawRect(...) / fillRect(...) - drawString(...) - setColor(...) / getColor()	
	Color	Constructeurs - Color(...)	
	Point	Constructeurs - Point(...) Méthodes - setLocation(...) / getLocation() - getX() / getY()	
java.util	ArrayList	Méthodes - add(...) - clear() - contains(...) - get(...) - indexOf(...) - remove(...) - set(...) - size() - 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(...) - indexOf(...) - 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()	