EXAMEN DE FIN D'ÉTUDES SECONDAIRES GÉNÉRALES

Sessions 2023 – QUESTIONNAIRE ÉCRIT

Date :	06.06.23	*Durée :*	14:15 - 17:15	*Numéro candidat :*	
Discipline :	Informatique	*Section(s) :*	GIG		

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 !

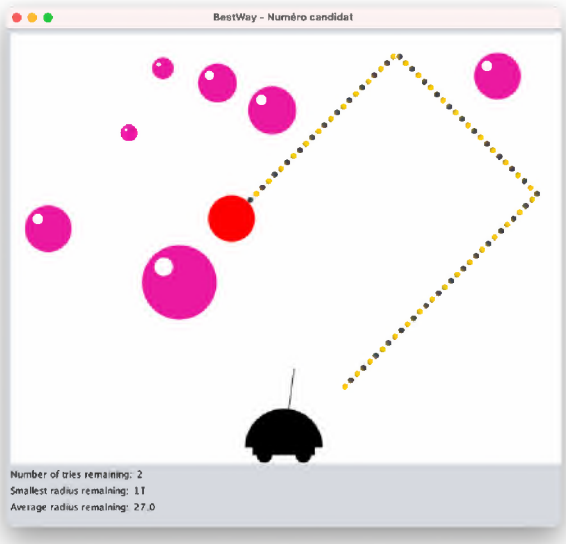
Vous trouvez une version exécutable du programme – **Demo.jar** – dans votre dossier.
Avant de commencer, il est recommandé de lancer et d'essayer ce programme.

Remarques générales :

- Ajoutez, tout en haut de chaque classe, votre code de l'examen en commentaire.
- Évitez les répétitions inutiles dans votre code source. Utilisez vos méthodes.
- Chaque classe fille doit, pour autant que possible, se servir des méthodes de sa classe mère.
- 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 de l'application « BestWay » — 60 points

Dans la suite, vous allez développer le jeu « BestWay ». Dans ce jeu, il s'agit d'éliminer, en trois essais au maximum, tous les obstacles avec un boulet rouge tiré par un canon dont le tube est en mouvement continu. Une pression du bouton gauche de la souris sur n'importe quelle position sur le terrain de jeu lance le boulet. Un essai se termine lorsque le boulet atteint le bord inférieur du terrain de jeu. Le canon peut être déplacé horizontalement en maintenant le bouton droit de la souris enfoncé tout en glissant la souris à gauche ou à droite sur le terrain de jeu. Une petite statistique informe le joueur de son progrès.

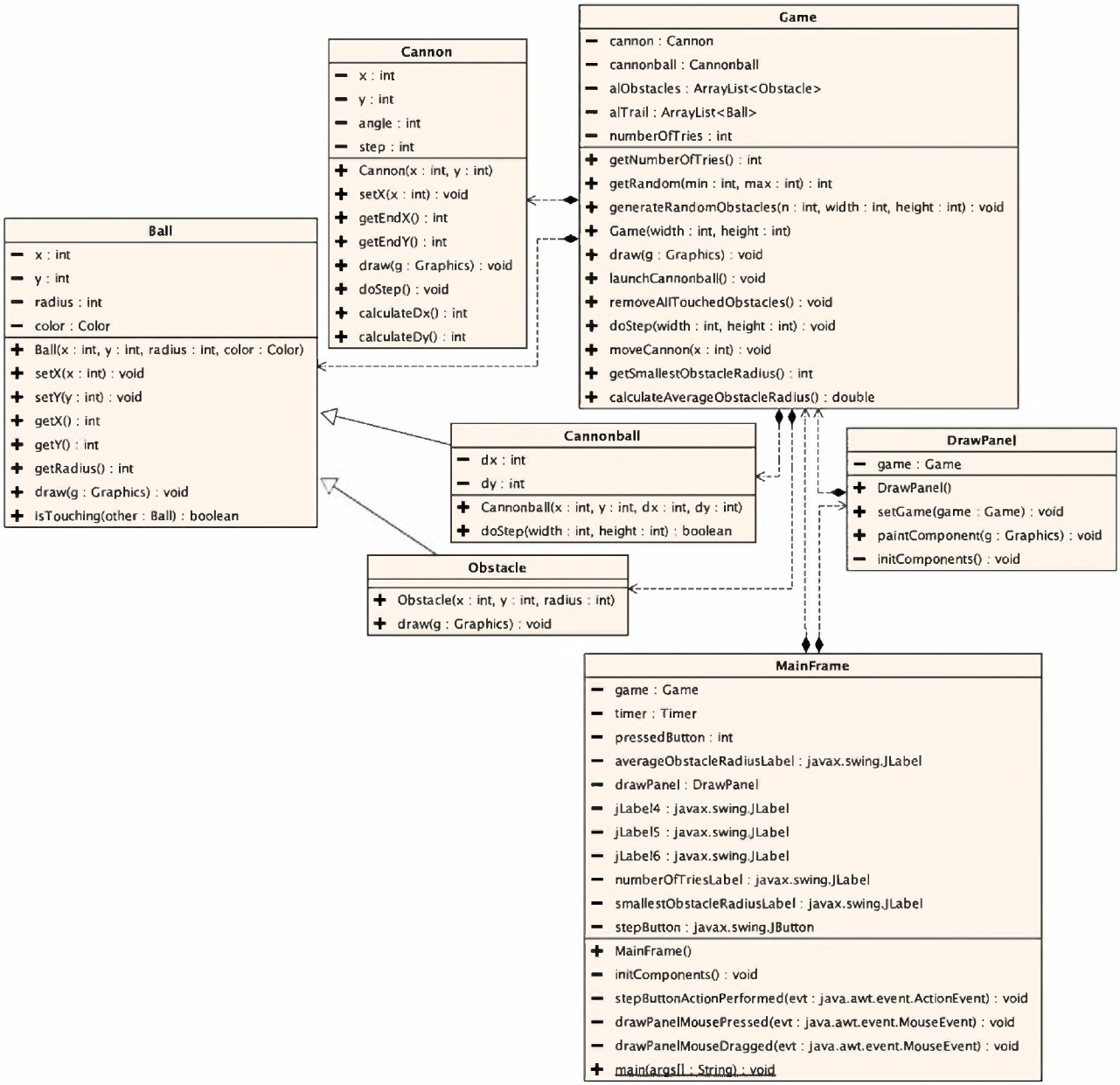


IMPORTANT :

Créez dans votre dossier un nouveau projet nommé « **BestWay** » et développez le projet en vous basant sur la version exécutable fournie, ainsi que sur le diagramme UML ci-dessous tout en respectant les instructions et précisions données dans la suite.

Toutes vos classes devront être sauvegardées à l'intérieur de ce projet.

Diagramme UML du programme « BestWay »



La classe Ball — 3,5 points

Implémentez la classe **Ball** – avec ses attributs, accesseurs et manipulateurs requis – en vous basant sur le diagramme UML donné (voir page 2) et les indications supplémentaires ci-dessous.

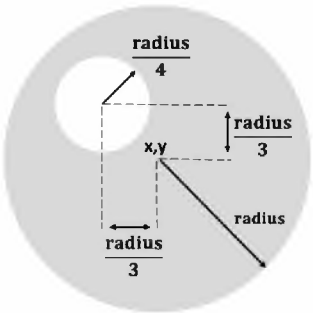
Cette classe représente un ballon de rayon **radius** et de couleur **color**. Les coordonnées **x** et **y** représentent le centre du ballon.

1. Le constructeur permet d'initialiser les attributs avec les valeurs passées en paramètre. **0,5p**
2. La méthode **draw** dessine le ballon sur le canevas. Le ballon est un simple disque rempli avec sa couleur (sans bordure). **1p**
3. La méthode **isTouching** retourne vrai si le ballon actuel touche un autre ballon passé en paramètre. Deux ballons se touchent si la distance de leur centres respectifs est inférieure ou égale à la somme de leurs rayons. **2p**

La classe Obstacle — 3 points

Implémentez la classe **Obstacle** en vous basant sur le diagramme UML donné (voir page 2) et les indications supplémentaires ci-dessous. Cette classe représente un obstacle.

1. Le constructeur permet d'initialiser les attributs avec les valeurs passées en paramètre. La couleur d'un obstacle est initialisée à la couleur avec le code RGB (235, 25, 160). **1p**
2. La méthode **draw** rajoute à l'objet dessiné un petit disque blanc de rayon $\frac{radius}{4}$ dont le centre est décalé de $\frac{radius}{3}$ vers la gauche et vers le haut par rapport au centre de l'obstacle. **2p**



La classe Cannonball — 3,5 points

Implémentez la classe **Cannonball** – avec ses attributs – en vous basant sur le diagramme UML donné (voir page 2) et les indications supplémentaires ci-dessous.

Cette classe représente le boulet sous forme de ballon rouge. Les attributs **dx** et **dy** définissent les pas de déplacement :

- **dx** positif ⇔ déplacement vers la droite,
- **dx** négatif ⇔ déplacement vers la gauche,
- **dy** positif ⇔ déplacement vers le bas,
- **dy** négatif ⇔ déplacement vers le haut.

1. Le constructeur permet d'initialiser les attributs. Le rayon est fixé à 30 pixels, la couleur est initialisée à la couleur rouge et **dx** et **dy** sont initialisés aux valeurs passées en paramètre. **0,5p**

2. La méthode **doStep** déplace le boulet. La largeur **width** et la hauteur **height** du terrain de jeu sont passées en paramètre. Le boulet doit rebondir au bord supérieur et aux bords à gauche et à droite et doit toujours rester complètement à l'intérieur du terrain de jeu. La méthode retourne vrai si le boulet atteint le bord inférieur du terrain de jeu, sinon la méthode retourne faux. **3p**

La classe Cannon — 8 points

Implémentez la classe **Cannon** – avec ses attributs et manipulateur requis – en vous basant sur le diagramme UML donné (voir page 2) et les indications supplémentaires ci-dessous.

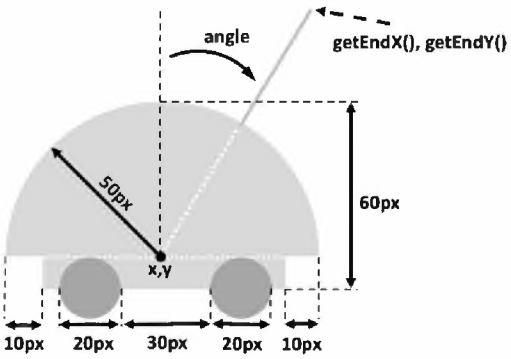
Cette classe représente le canon qui permet de lancer le boulet et dispose des attributs suivants :

x, y les coordonnées du centre du demi-disque de la coupole,

angle l'angle d'inclinaison (en degrés) du tube du canon,

step le pas de rotation (en degrés) du tube :

- **step** positif ⇔ rotation vers la droite,
- **step** négatif ⇔ rotation vers la gauche.



1. Le constructeur permet d'initialiser les attributs **x** et **y** par les valeurs des paramètres. L'angle est initialisé à zéro et **step** à la valeur 1. **0,5p**

2. Les méthodes **getEndX** et **getEndY** calculent et retournent les coordonnées du point supérieur du tube du canon (voir image ci-dessus). Pour faire ceci, copiez simplement les lignes de code suivantes dans le corps respectif de chaque méthode :

   - pour la méthode **getEndX** ➔ **return** `(int)(x+100*Math.sin(angle*Math.PI/180));`
   - pour la méthode **getEndY** ➔ **return** `(int)(y-100*Math.cos(angle*Math.PI/180));`

   0,5p

3. La méthode **draw** dessine le canon sur le canevas. Le canon est composé d'un demi-disque noir qui représente la coupole et d'un rectangle noir qui forme la base du canon. Les deux roues de diamètre 20 pixels touchent la coupole. Le tube du canon est représenté par une ligne noire dont l'origine se trouve à la position **x**,**y** (voir aussi l'image ci-dessus). **4p**

4. La méthode **doStep** incrémente ou décrémente l'angle du tube de **step** degrés selon la direction actuelle tout en veillant que l'angle reste toujours dans l'intervalle [-50, +50] degrés. Si une de ces limites est atteinte, le mouvement est inversé. **2p**

5. Les méthodes **calculateDx** et **calculateDy** (utilisées dans la classe **Game** lors d'un tir du boulet) retournent le résultat des expressions suivantes :
   - **calculateDx** retourne le résultat de $\frac{getEndX() - x}{8}$,
   - **calculateDy** retourne le résultat de $\frac{getEndY() - y}{8}$. **1p**

La classe Game — 32 points

Implémentez la classe **Game** – avec ses attributs et accesseur requis – en vous basant sur le diagramme UML donné (voir page 2) et les indications supplémentaires ci-dessous.

Cette classe est responsable de la gestion du jeu avec ses attributs :

cannon	le canon,
cannonball	le boulet qui est lancé par le canon,
alObstacles	la liste avec tous les obstacles,
alTrail	la liste de balles qui tracent la trajectoire du boulet,
numberOfTries	le nombre d'essais restants. **0,5p**

1. La méthode **getRandom** retourne un nombre aléatoire entier ϵ **[min, max]**. **1p**

2. La méthode **generateRandomObstacles** complète la liste des obstacles de façon à contenir **n** obstacles aléatoires selon les règles ci-dessous.
   - Le rayon des obstacles varie entre 10 et 50 pixels (bornes comprises).
   - À l'horizontale, les obstacles doivent être répartis de manière aléatoire sur toute la largeur du terrain de jeu de largeur **width** sans dépasser les bords latéraux.
   - À la verticale, les obstacles doivent se trouver entièrement dans les deux tiers supérieurs du terrain de jeu de hauteur **height**.
   - Un obstacle ne doit pas en toucher un autre. **6p**

3. Le constructeur initialise le nombre d'essais à 3. Ensuite, il crée un nouveau canon au milieu et positionné à 20 pixels au-dessus (all. : *oberhalb*) du bord inférieur du terrain de jeu de largeur **width** et de hauteur **height**. Puis, le constructeur crée 8 obstacles. **1,5p**

4. La méthode **draw** dessine toutes les balles de la liste **alTrail**, les obstacles, le canon et, si possible, le boulet. **2p**

5. La méthode **launchCannonball** effectue les étapes suivantes uniquement si le boulet n'existe pas :
   - le nombre d'essais est décrémenté ;
   - la liste **alTrail** est vidée ;
   - un nouveau boulet est créé à la position supérieure du tube du canon. Les valeurs pour **dx** et **dy** peuvent être calculées directement à partir des méthodes **calculateDx** et **calculateDy** du canon. **2,5p**

6. La méthode **removeAllTouchedObstacles** supprime tous les obstacles qui touchent le boulet. **3p**

7. La méthode **doStep** effectue les étapes suivantes :
   - elle avance d'un pas l'angle du tube du canon ;
   - si le boulet existe, la méthode effectue les étapes suivantes :
     - elle avance le boulet d'un pas ;
     - si le boulet n'a pas encore atteint le bord inférieur du terrain de jeu, une nouvelle balle de rayon 4 pixels sera ajoutée à la liste **alTrail** aux mêmes coordonnées que le boulet. La couleur de cette nouvelle balle doit être choisie de sorte que la couleur des balles alterne entre l'orange et le gris foncé. Ensuite, la méthode supprime tous les obstacles qui touchent le boulet ;
     - si, en revanche, le boulet a atteint le bord inférieur du terrain de jeu, le boulet a terminé sa trajectoire et l'attribut **cannonball** est mis à **null**. Lorsqu'il ne reste plus d'obstacles ou que le joueur n'a plus d'essais à sa disposition, le jeu est relancé, c.-à-d. le nombre d'essais est ajusté à 3 et la liste des obstacles est complétée de sorte qu'elle contienne de nouveau 8 obstacles. **6p**

8. La méthode **moveCannon** fixe la position horizontale du canon à la valeur passée en paramètre. **0,5p**

9. La méthode **getSmallestObstacleRadius** retourne le rayon de l'obstacle le plus petit, ou -1 si la liste est vide. **5p**

10. La méthode **calculateAverageObstacleRadius** retourne le rayon moyen de tous les obstacles, ou zéro si la liste est vide. **4p**

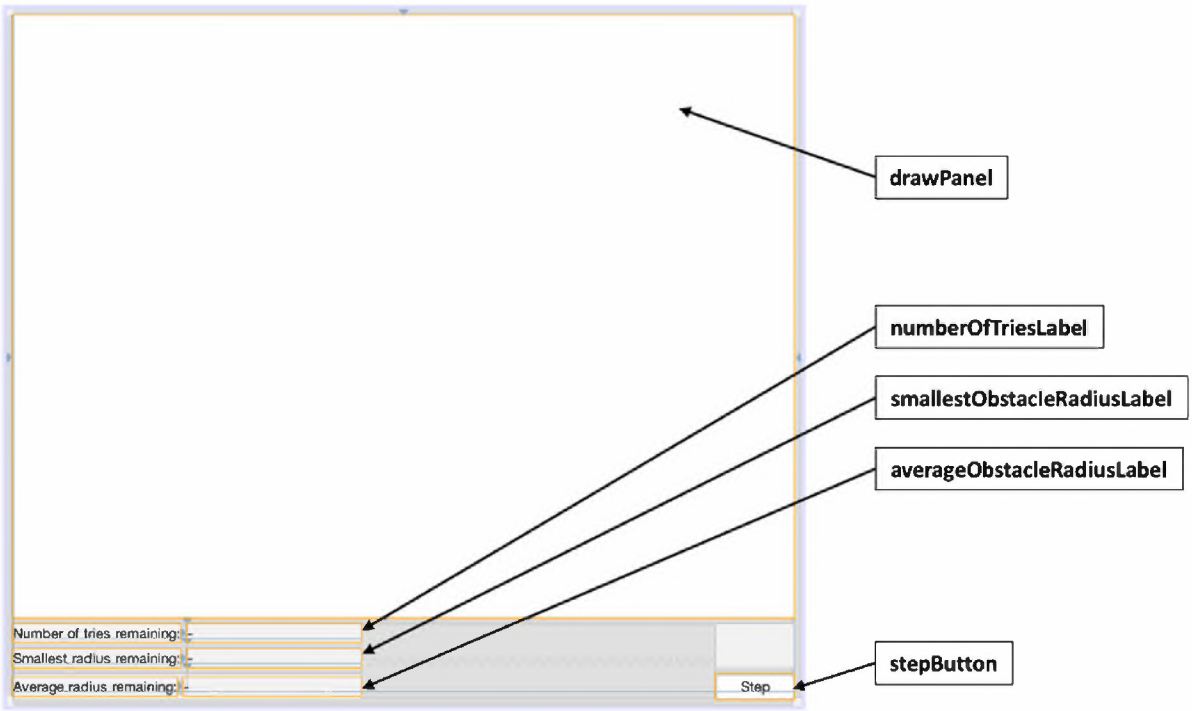
La classe DrawPanel — 1 point

Implémentez la classe **DrawPanel** – avec son attribut et son manipulateur requis – en vous basant sur le diagramme UML donné (voir page 2) et l'indication supplémentaire ci-dessous.
La méthode **paintComponent** dessine, si possible, le jeu **game** sur un fond blanc.

La classe MainFrame — 9 points

Implémentez la classe **MainFrame** – avec ses attributs requis – en vous basant sur le diagramme UML donné (voir page 2) et les indications supplémentaires ci-dessous.

1. Reproduisez fidèlement l'interface graphique de la classe **MainFrame**, illustrée ci-dessous et respectez la nomenclature indiquée. À titre d'indication, la fiche principale a une largeur de 730 pixels et une hauteur de 640 pixels.
   Ajoutez le titre **« BestWay – »** suivi de votre numéro de candidat à la fiche principale. **1p**



2. Effectuez les initialisations nécessaires. Le chronomètre exécute 40 fois par seconde la méthode du bouton caché **stepButton**. Ce chronomètre est lancé au démarrage du programme. **3p**

3. Le bouton **stepButton** avance d'un pas le jeu et actualise l'interface graphique y compris les trois libellés. **2p**

4. Si le joueur enfonce un bouton de la souris sur le canevas, le numéro du bouton enfoncé est sauvegardé dans l'attribut **pressedButton**. Si le bouton gauche de la souris a été enfoncé, un nouveau boulet est lancé. **2p**

5. Si le joueur déplace le curseur de la souris sur le canevas en gardant le bouton droit enfoncé, le canon est positionné à l'abscisse de la souris. **1p**

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()	