Code branche **INFOR**	**Ministère de l'Education nationale, de l'Enfance et de la Jeunesse** EXAMEN DE FIN D'ETUDES SECONDAIRES TECHNIQUES Régime technique – **Session 2013/2014**	
Épreuve **écrite**	*Branche*	*Division / Section*
Durée épreuve **3 h**	**Informatique** **Java**	**G E**
Date épreuve 28/05/2014		

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 : **LXY_GE1_13**).

*Tous vos fichiers devront être sauvegardés à l'intérieur de ce sous-dossier, qui sera appelé **'votre dossier'** dans la suite !*

Question 1 — *22 points*

Ouvrez le projet ***Formula1Race*** de votre dossier.

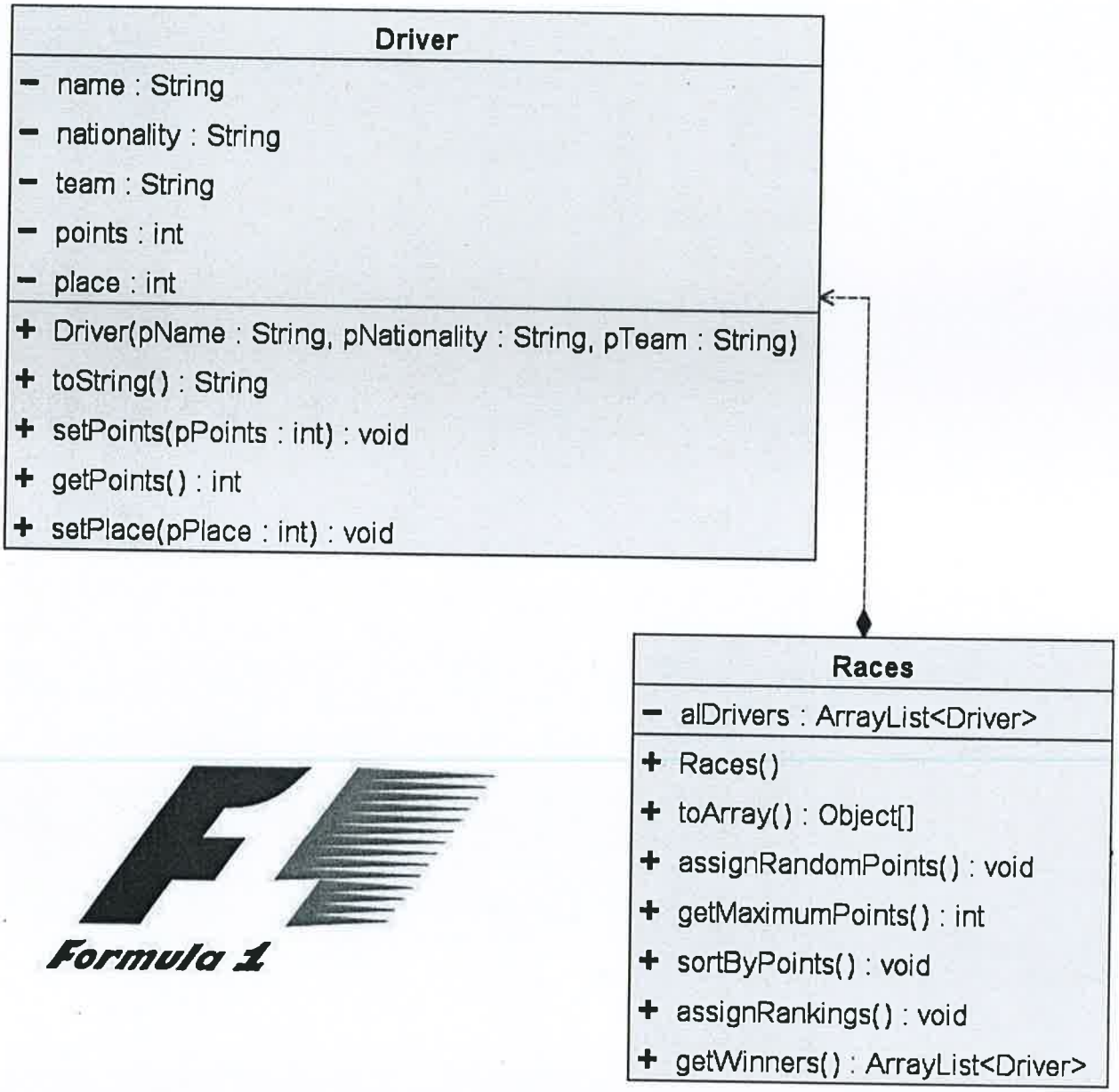
- l'environnement de programmation est à votre choix (**Unimozer** ou **NetBeans**),
- pour tester vos méthodes modifiées ouvrez le projet dans l'environnement **NetBeans**. L'interface graphique est déjà programmée.

Le programme sert à simuler les résultats de courses de Formule 1.

Vous trouvez une version exécutable du programme (**Formua1Race.jar**) dans votre dossier. Avant de continuer, il est recommandé de lancer et de tester ce programme.

On considère les classes **Driver** et **Races** dont la description **UML** (à la page suivante) et le code **JAVA** sont fournis.

Diagramme U M L :



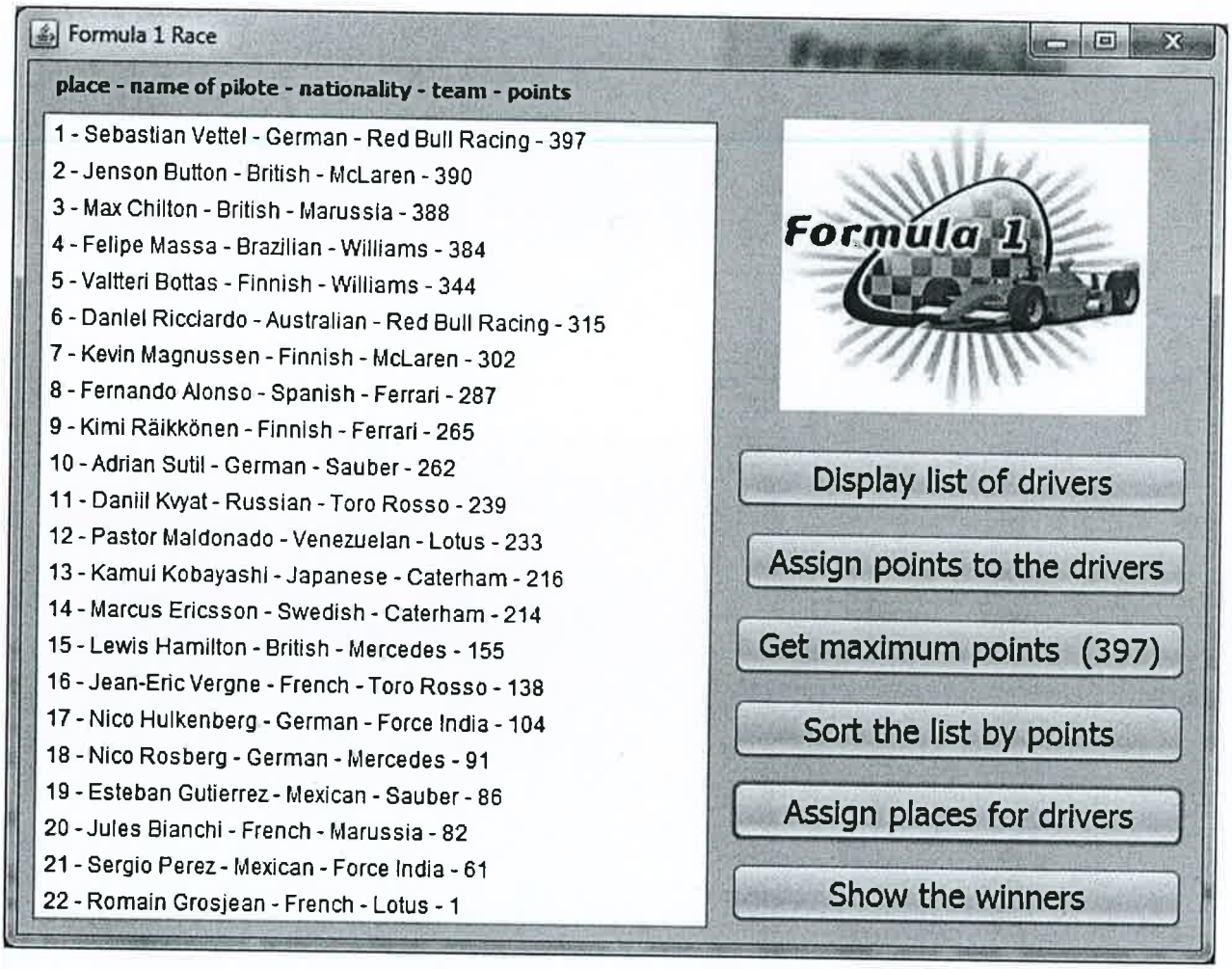
Consignes et informations générales :

- la classe **Driver** doit être utilisée mais ne doit pas être modifiée, les attributs d'un pilote de course sont son nom, sa nationalité, son équipe, les points obtenus dans les courses et sa place au classement final.

- les 6 méthodes suivantes de la classe **Races** doivent être complétées :
    - **toArray ( )**
    - **assignRandomPoints ( )**
    - **getMaximumPoints ( )**
    - **sortByPoints ( )**
    - **assignPlaces ( )**
    - **getWinners ( )**

Une interface graphique pourrait par exemple se présenter comme suit :

Vous n'avez PAS besoin de définir une interface graphique !



Travail à réaliser :

1. Écrivez la méthode **toArray ( )** qui transforme la liste **alDrivers** en Object [ ] afin de pouvoir l'afficher dans une JList. *(1 point)*

2. Écrivez la méthode **assignRandomPoints ( )** pour attribuer des points aléatoires aux pilotes de course. Il s'agit du nombre de points que les pilotes ont obtenus dans l'ensemble des courses. Ici on va donner des points aléatoires entre **0** et **400** aux pilotes.

   (3 points)

3. Écrivez la méthode **getMaximumPoints ( )** qui détermine et retourne le maximum des points de la liste **alDrivers**. La déclaration de la méthode et l'instruction **return** sont déjà programmées. Insérez votre code à l'endroit prévu et adaptez l'instruction **return** selon vos besoins! *(4 points)*

4. Écrivez la méthode **sortByPoints ( )** qui trie les pilotes de course de la liste **alDrivers** par **ordre décroissant** de leur points à l'aide de l'algorithme de **tri par sélection directe**. La déclaration de la méthode est déjà programmée. Insérez votre code à l'endroit prévu! *(8 points)*

5. Écrivez la méthode **assignRankings ( )** qui attribue les places obtenues aux pilotes après que la liste soit triée. On suppose que la liste est déjà triée avant l'appel de **assignRankings ( )**. *(3 points)*

6. Écrivez la méthode **getWinners ( )** qui retourne la liste des **3 premiers** pilotes de course au classement final. La liste originale doit rester inchangée. La déclaration de la méthode est déjà programmée. Insérez votre code à l'endroit prévu! On suppose que la liste est déjà triée avant l'appel de **getWinners ( )**. *(3 points)*

Question 2 — *38 points*

Dans la suite, vous allez développer un petit programme en **NetBeans** pour réaliser une animation avec des balles de différentes couleurs. Il y a deux sortes de balles, les unes deviennent plus grandes jusqu'à ce qu'elles atteignent leur rayon maximal, les autres deviennent plus petites jusqu'à ce que leur rayon devienne zéro. Dans les deux cas elles disparaissent alors de l'animation.

Vous trouvez une version exécutable du programme (**Balls.jar**) dans votre dossier. Avant de continuer, il est recommandé de lancer et de tester ce programme. Testez les fonctions des deux boutons (gauche et droite) de votre souris sur le panneau.

Créez avec **NetBeans** un nouveau projet nommé **Balls** dans votre dossier.

Réalisez ce programme en vous basant sur la version exécutable fournie ainsi que sur le diagramme **UML** (page suivante) tout en respectant les instructions et précisions données dans la suite.

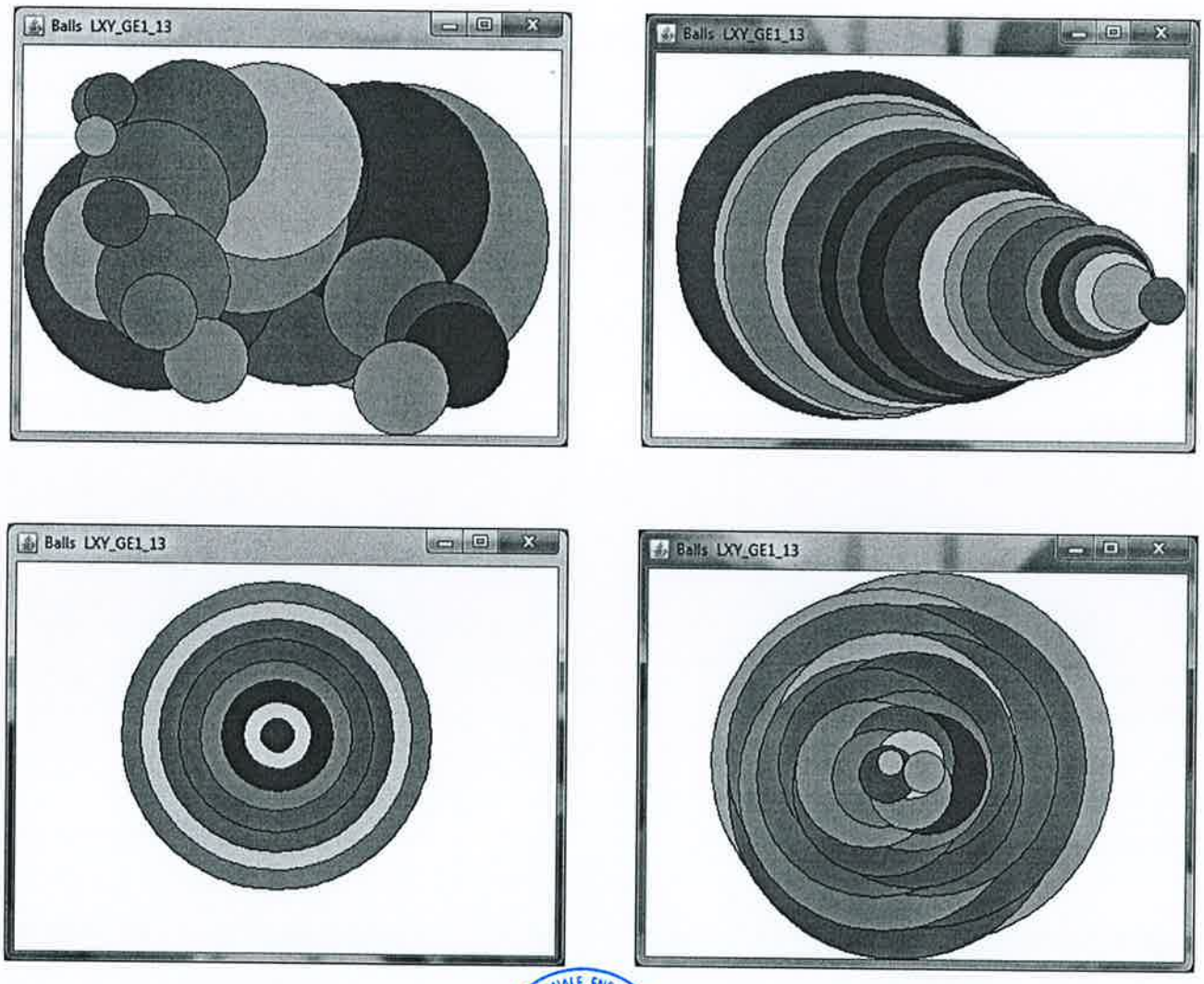
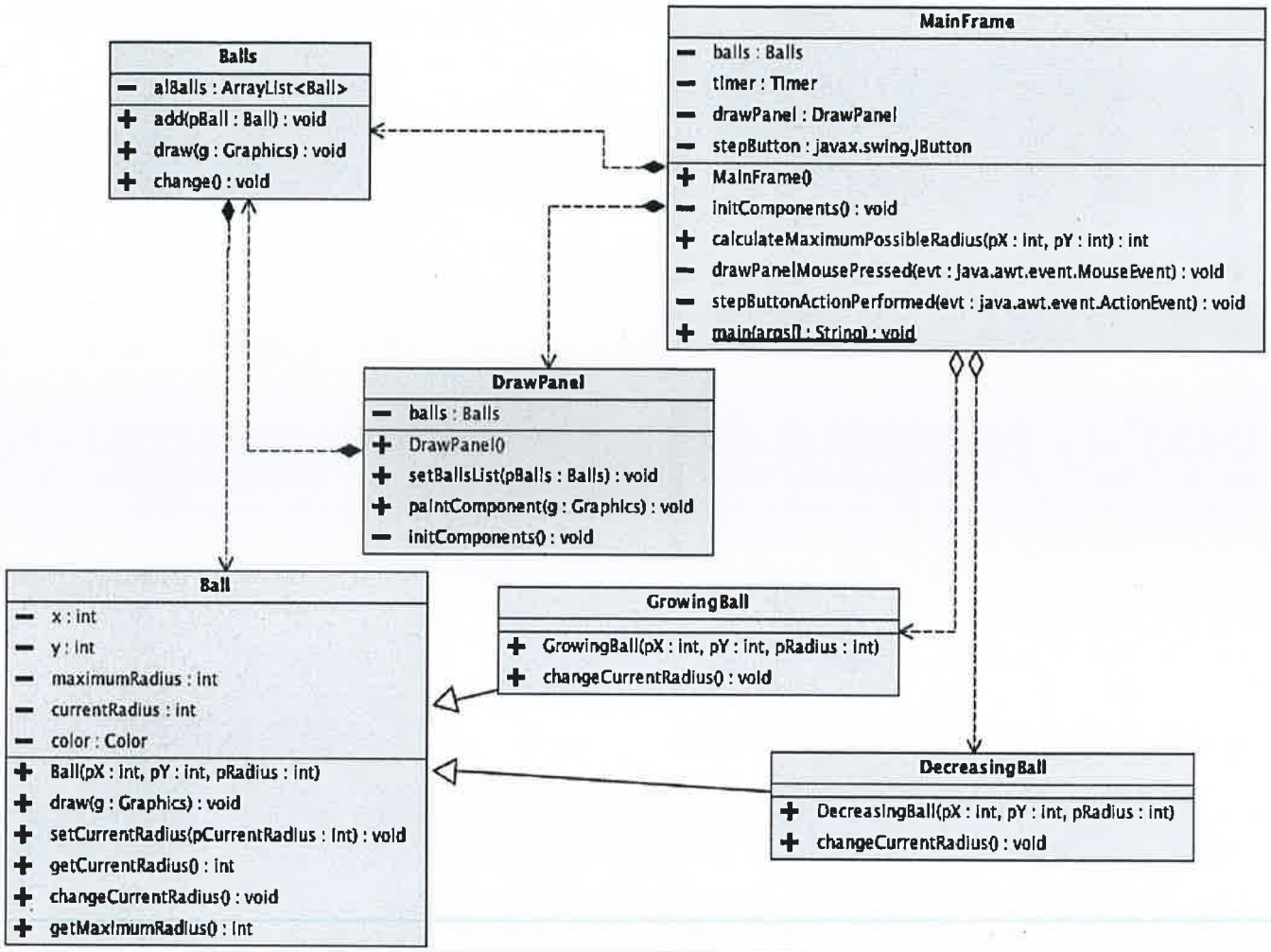


Diagramme U M L :



Travail à réaliser :

1. Écrivez le code source JAVA de la classe **Ball**. *(9 points)*

   - Le **constructeur** initialise les coordonnées x et y du **centre** de la balle ainsi que le rayon maximal et génère une **couleur** aléatoire pour la balle (R, G et B auront des valeurs aléatoires entre **0** et **255**).
   - La méthode **draw (Graphics g)** dessine la balle dans sa couleur et le bord est dessiné avec une ligne noire.
   - La méthode **setCurrentRadius (...)** initialise le rayon actuel de la balle.
   - La méthode **getCurrentRadius (...)** retourne le rayon actuel de la balle.
   - La méthode **changeCurrentRadius ( )** ne fait rien, donc mettez comme commentaire dans le corps cette méthode : ***Méthode vide***.
   - La méthode **getMaximumRadius ( )** retourne le rayon maximal de la balle.

2. Écrivez le code source JAVA de la classe **Balls**. *(7 points)*

   - La méthode **addBall (…)** ajoute une balle à la fin de la liste.
   - La méthode **draw (Graphics g)** dessine toutes les balles de la liste.
   - La méthode **change ( )** change le rayon actuel des balles. Si le rayon actuel dépasse le rayon maximal ou s'il devient zéro alors cette balle est supprimée dans la liste. Utiliser une boucle à rebours (Boucle for à l'envers).

3. Écrivez le code source JAVA de la classe **GrowingBall**. *(3 points)*

   - La classe GrowingBall est une classe fille de la classe Ball
   - Le **constructeur** fait appel au **constructeur hérité** pour initialiser x et y ainsi que le rayon maximal de la balle et initialise le rayon actuel avec la valeur 1.
   - La méthode **changeCurrentRadius ( )** **incrémente** le rayon actuel de 1.

4. Écrivez le code source JAVA de la classe **DecreasingBall**. *(3 points)*

   - La classe DecreasingBall est une classe fille de la classe Ball
   - Le **constructeur** fait appel au **constructeur hérité** pour initialiser x et y ainsi que le rayon maximal de la balle et initialise le rayon actuel avec la valeur du **rayon maximal**.
   - La méthode **changeCurrentRadius ( )** **décrémente** le rayon actuel de 1.

5. Écrivez le code source JAVA de la classe **DrawPanel**. *(3 points)*

   - La méthode **setBallsList (…)** initialise la liste avec la liste passée comme paramètre.
   - La méthode **paintComponent(Graphics g)** dessine un rectangle blanc sur toute la surface du drawPanel et dessine toutes les balles de la liste si la liste n'est pas vide.

6. Écrivez le code source JAVA de la classe **MainFrame**. *(13 points)*

- Le constructeur **MainFrame ( )**
  - met comme titre de la fenêtre principale

    Balls LXY_GE1_13 *(avec vos données)*
  - passe la liste **balls** au drawPanel
  - crée un nouveau chronomètre **timer** avec une périodicité de 50 ms
  - démarre le chronomètre
  - rend invisible le bouton stepButton

- La méthode **calculateMaximumPossibleRadius (. . .)** détermine le rayon maximal possible de la balle sans qu'elle ne dépasse la surface visible du drawPanel. Le rayon maximal correspond donc à la plus petite distance du centre de la balle à un bord du drawPanel. Pour calculer cela, la méthode détermine les 4 distances du centre de la balle jusqu'aux 4 bords et retourne le minimum de ces distances comme rayon maximal du cercle.

- La méthode **drawPanelMousePressed (. . .)**
  - crée une nouvelle balle
    - ➢ le **centre** de la balle est la position où on a enfoncé un bouton de la souris sur le drawPanel
    - ➢ le rayon de la balle est déterminé avec la méthode **calculateMaximumPossibleRadius (...)**
    - ➢ crée une nouvelle balle du type **GrowingBall** si on a enfoncé le bouton **gauche** de la souris ou une balle du type **DecreasingBall** si on a enfoncé le bouton **droit** de la souris
    - ➢ ajoute cette nouvelle balle dans la liste

- La méthode **stepButtonActionPerformed ( )**
  - change le rayon de toutes les balles de la liste
  - redessine le drawPanel

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() - 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()	