Contrôle 4: Informatique Solution 14 juin 2017

Cours de mathématiques spéciales Semestre de printemps

(écrire lisiblement et avec au maximum trois couleurs différentes s.v.p.)

Nom : ……..

Prénom : ……..….

Question	Barème	Points
1	136	
2	64	
Total	200	

Note :

Indications générales

- Durée de l'examen : **105 minutes**.
- Posez votre **carte d'étudiant** sur la table.
- La réponse à chaque question doit être rédigée **à l'encre** sur la place réservée à cet effet à la suite de la question.
  Si la place prévue ne suffit pas, vous pouvez demander des feuilles supplémentaires aux surveillants ; chaque feuille supplémentaire doit porter **nom, prénom, n° du contrôle, branche et date**. Elle ne peut être utilisée que pour **une seule question**.
- Les feuilles de brouillon ne sont pas à rendre ; elles ne seront **pas corrigées** ; des feuilles de brouillon supplémentaires peuvent être demandées en cas de besoin auprès des surveillants.
- Les feuilles d'examen doivent être rendues **agrafées**.

Indications spécifiques

- Toutes les questions qui suivent se réfèrent au langage de programmation **Java** (à partir du **JDK 8.0**).
- A part les 3 polycopiés du cours et les notes des 3 derniers cours (à savoir les Chapitres 22, 23 et 24), **aucune** autre source bibliographique n'est autorisée et ne doit donc être consultée durant le contrôle.

Remarques initiales :

- Il est conseillé de lire chaque question jusqu'à la fin, avant de commencer la rédaction de la solution.
- Suite à une instruction "***return***", une méthode (qui retourne un résultat ou qui est de type ***void***) est quittée "immédiatement".
- Soit (dans le plan) un segment de droite qui "part" d'un point de coordonnées (x_A, y_A) et "arrive" en un point de coordonnées (x_B, y_B). Si on divise ce segment en n intervalles égaux dont les extrémités sont numérotées à l'aide d'un compteur k qui varie de 0 à n, les coordonnées d'un point k sont $(x_k = x_A + k*(x_B - x_A)/n,\ y_k = y_A + k*(y_B - y_A)/n)$.
- Afin de savoir si une animation (graphique) est en cours ou pas, on peut appeler la méthode d'instance sans argument ***isRunning()*** pour le moteur de l'animation car elle retourne la valeur booléenne correspondant à **vrai** si l'animation est en cours ou la valeur booléenne correspondant à **faux** autrement.

Question 1

Le but de cet exercice est de permettre à l'utilisateur d'interagir avec une interface graphique (GUI) qui sera décrite par la suite. Plus précisément, au début de l'exécution du projet à réaliser, une interface graphique comme celle présentée dans la **Figure 1** doit apparaître dans le coin supérieur gauche de l'écran. Cette GUI correspond à un container de premier niveau avec 2 "régions" :

- la région supérieure de couleur verte, mais apparaissant comme grise foncée dans la copie d'écran, affiche au centre un mobile (initialement figé) en forme de disque (cercle plein) de couleur rouge ;
- la région inférieure de couleur jaune, mais apparaissant comme grise claire dans la copie d'écran, est munie de deux cases à cocher (initialement non sélectionnées) avec les textes associés ***Cercle*** et ***Carré***.

Quand l'utilisateur clique (avec la souris) à un endroit quelconque de la région supérieure, le mobile se déplace en ligne droite à partir de l'endroit où il se trouvait au moment du clic et jusqu'au point où se trouvait le pointeur de la souris au moment du clic.

Durant ce déplacement tout nouveau clic de souris reste sans effet et la couleur du mobile change successivement entre rouge et bleu.

Une fois le centre de symétrie du mobile arrivé au point où se trouvait le pointeur de la souris au moment du clic qui a déclenché le mouvement, le mobile s'arrête. Ensuite, l'utilisateur peut faire un nouveau clic (dans la région supérieure) afin de déplacer le mobile vers une nouvelle position.

De plus, à l'aide de deux cases à cocher, l'utilisateur peut choisir la forme du mobile, à savoir soit un disque soit un carré plein.

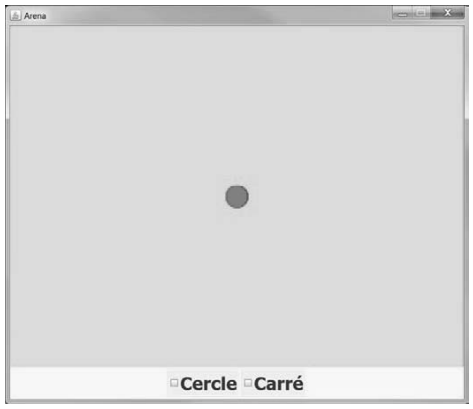

Figure 1

On considère un projet Java qui implémente les consignes mentionnées ci-dessus et qui est muni d'un package nommé ***cms_ctr4*** contenant deux classes publiques :

- la classe graphique ***Arena*** qui correspond à l'interface graphique décrite auparavant ;
- la classe principale ***CP_Ctr4Exo1*** dont le code source n'est pas demandé et qui contient la méthode publique et statique ***main*** qui crée une instance de la classe graphique ***Arena***.

Dans cet exercice, on vous demande d'écrire le code source de la classe graphique ***Arena*** en fonction des indications données ci-dessous.

1.1 La classe ***Arena*** est publique, appartient au package ***cms_ctr4*** et correspond à un container de premier niveau (top level container).

A part plusieurs champs privés, cette classe contient un constructeur sans argument et les définitions de trois classes internes et privées, à savoir :

- ***Surface*** qui correspond à un container intermédiaire dont une instance est placée dans la région supérieure du containeur de premier niveau et qui représente la zone de dessin où on affiche l'animation graphique ;
- ***Observateur*** qui (sans être une classe graphique) permet la création d'un écouteur (non graphique) des événements liés à la souris et au moteur d'animation ;
- ***PanCheck*** qui correspond à un container intermédiaire dont une instance est placée dans la région inférieure du container de premier niveau et qui sert à abriter deux cases à cocher qu'il écoute lui-même.

Les significations des champs de la classe ***Arena*** sont les suivantes:

- ***large*** et ***haut*** correspondent, respectivement, à la largeur et à la hauteur (en pixels) de la région supérieure ;
- ***rayon*** correspond soit au rayon du disque soit à la moitié du côté du carré plein utilisés pour représenter le mobile ;
- ***xIn*** et ***yIn*** correspondent aux coordonnées du centre de symétrie du mobile avant qu'il commence à bouger ;
- ***x*** et ***y*** correspondent aux coordonnées du centre de symétrie du mobile à un moment courant durant son mouvement (donc durant l'animation) ;
- ***xFin*** et ***yFin*** correspondent à la fois aux coordonnées de la position du pointeur de la souris au moment du clic qui déclenche le mouvement du mobile et aux coordonnées de la position "finale" du mobile ;
- ***delta*** correspond à la durée du mouvement du mobile (en millisecondes) ;
- ***nb*** correspond au nombre de positions intermédiaires du mobile durant une animation ;
- ***i*** correspond à un compteur qui compte le nombre de "pas" effectués par le mobile durant son mouvement ;
- ***obs*** correspond à l'écouteur de la souris et du moteur de l'animation, et est de type ***Observateur*** ;
- ***surf*** correspond à une instance de la classe interne ***Surface*** ;
- ***pc*** correspond à une instance de la classe interne ***PanCheck*** ;
- ***forme*** correspond à une chaine de caractères qui sera **Cercle** ou **Carré** et qui indique la forme du mobile ;
- ***moteur*** correspond au moteur de l'animation.

Dans la définition du constructeur sans argument et sans modificateur d'accès, vous devez :

- prévoir une instruction qui impose l'arrêt de la machine virtuelle suite à la fermeture du container de premier niveau, container appelé par la suite tout simplement la fenêtre ;
- rendre la fenêtre visible ;
- rendre la fenêtre non redimensionnable ;
- donner le titre ***Arena*** à la fenêtre ;
- ajouter le container intermédiaire ***surf*** dans la région "supérieure" de la fenêtre ;
- ajouter le container intermédiaire ***pc*** dans la région "inférieure" de la fenêtre ;
- appeler une méthode qui assure que la taille de la fenêtre s'adapte (à l'exécution et de manière optimale) à son contenu.

Selon les consignes données, écrivez ci-dessous le début du code source de la classe graphique ***Arena*** (c'est-à-dire tout le code avant les définitions des trois classes internes).

```
//Déclaration du package
package cms_ctr4;

//Imports des packages nécessaires
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;


//En-tête de la classe englobante Arena
public class Arena extends JFrame {
{
	//Déclaration des champs privés
	private int large = 800, haut = 600;
	private int rayon = 20;
	private int xIn, yIn, x, y, xFin, yFin;
	private int delta = 1000;
	private int nb = 500;
	private int i = 0;
	private Observateur obs = new Observateur();
	private Surface surf = new Surface();
	private PanCheck pc = new PanCheck();
	private String forme = "Cercle";
	//on déclare un champ de type "moteur d'animation" qui envoie nb
	//signaux durant le temps delta à l'intention de l'écouteur obs
	private Timer moteur = new Timer(delta/nb, obs);
```



```
	//Définition (en-tête et corps) du constructeur
	Arena() {

		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		setVisible(true);

		setResizable(false);

		setTitle("Arena");

		add(surf, BorderLayout.NORTH);

		add(pc, BorderLayout.SOUTH);

		pack();

	}
```

1.2 La classe interne ***Surface*** est privée et correspond à un container intermédiaire (qui est affiché dans la région supérieure du container de premier niveau). Elle n'a aucun champ propre mais définit un constructeur sans argument et redéfinit la méthode ***paint()***.

Dans la définition du constructeur sans argument et sans modificateur d'accès, vous devez :

- indiquer que le container intermédiaire qui est en train d'être créé doit avoir une taille préférée de largeur et hauteur données (respectivement) par les champs ***large*** et ***haut*** ;
- donner la couleur verte ("green" en anglais) comme couleur de fond du conteneur ;
- donner aux champs ***x*** et ***xIn*** une même valeur correspondant à l'abscisse du centre du container ;
- donner aux champs **y** et ***yIn*** une même valeur correspondant à l'ordonnée du centre du container ;
- ajouter le champ ***obs*** comme écouteur des événements de bas niveau liés à la souris.

Dans la redéfinition de la méthode ***paint( )***, vous devez :

- appeler la méthode d'origine (de la classe mère) ;
- d'un appel à l'autre de cette méthode, changer la couleur de dessin entre rouge ("red" en anglais") et bleu ("blue" en anglais) (en tirant profit du fait que les appels successifs de cette méthode correspondent en fait à des valeurs successives du compteur ***i*** qui change donc de parité d'un appel à l'autre) ;
- si le champ ***forme*** correspond à la constante littérale de type chaîne de caractères (donc au texte) **Cercle**, dessiner un disque (cercle plein) ayant les coordonnées de son centre données par les valeurs des champs ***x*** et ***y*** et le rayon donné par la valeur du champ ***rayon*** ;
- autrement, si le champ ***forme*** correspond à la constante littérale de type chaîne de caractères **Carré**, dessiner un carré plein ayant les coordonnées de son centre de symétrie données par les valeurs des champs ***x*** et ***y*** et le rayon donné par la valeur du champ ***rayon***.

Ecrivez à la page suivante la définition de la classe interne ***Surface*** (c'est-à-dire son en-tête et son corps contenant la définition du constructeur et la redéfinition de la méthode ***paint()***).

X


```
private class Surface extends JPanel {

    Surface() {

        setPreferredSize(new Dimension(large, haut));
        setBackground(Color.GREEN);
        x = xIn = large/2;
        y = yIn = haut/2;
        addMouseListener(obs);
    }//fin du constructeur

    @Override
    public void paint(Graphics g) {

        super.paint(g);

        if(i % 2 == 0) {
            g.setColor(Color.RED);
        }else {
            g.setColor(Color.BLUE);
        }
        if(forme.equals("Cercle")) {
            g.fillOval(x-rayon, y-rayon,
                       2*rayon, 2*rayon);
        }else if(forme.equals("Carré")) {
            g.fillRect(x-rayon, y-rayon,
                       2*rayon, 2*rayon);
        }
    }//fin de la méthode paint
}//fin de la classe interne Surface
```

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

1.3 La classe interne ***Observateur*** est privée et permet la création du champ ***obs*** (de la classe englobante) afin d'écouter à la fois les événements de bas niveau liés aux actions de la souris (dans le container intermédiaire placé dans la région supérieure du container de premier niveau) et les événements sémantiques (ou abstraits de haut niveau) produits par le moteur de l'animation. Par conséquent, cette classe ne contient que les redéfinitions des gestionnaires d'événements appropriés.

Dans la redéfinition du "bon" gestionnaire qui traite les événements de bas niveau liés à la souris, vous devez procéder ainsi :

- si une animation est en cours (voir l'indication donnée tout au début), quitter immédiatement la méthode ;
- (autrement), démarrer le moteur de l'animation ;
- récupérer l'abscisse de la position du pointeur de la souris au moment du clic et stocker-la dans le champ ***xFin*** ;
- récupérer l'ordonnée de la position du pointeur de la souris au moment du clic et stocker-la dans le champ ***yFin***.

Dans la redéfinition du gestionnaire qui traite les événements sémantiques liés au moteur de l'animation, vous devez procéder ainsi :

- si la source de l'événement n'est pas le moteur de l'animation, quitter immédiatement la méthode ;
- (autrement) incrémenter le champ ***i*** qui compte les positions intermédiaires du mobile ;
- si le mobile n'est pas encore arrivé à la position finale :
    - calculer l'abscisse de la nouvelle position intermédiaire du mobile (voir l'indication donnée tout au début) et stocker-la dans le champ ***x*** ;
    - calculer l'ordonnée de la nouvelle position intermédiaire du mobile (voir l'indication donnée tout au début) et stocker-la dans le champ ***y*** ;
- autrement :
    - copier la valeur du champ ***xFin*** dans le champ ***xIn*** ;
    - copier la valeur du champ ***yFin*** dans le champ ***yIn*** ;
    - remettre le compteur pour les positions intermédiaires ***i*** à zéro ;
    - arrêter le moteur de l'animation ;
- exiger que l'interface graphique tienne compte des calculs que vous venez de faire.

Ecrivez ci-dessous la définition de la classe interne ***Observateur*** (c'est-à-dire son en-tête et son corps contenant les redéfinitions de tous les gestionnaires d'événements).


```
private class Observateur implements MouseListener,
                                     ActionListener {
    @Override
    public void mouseClicked(MouseEvent me) {
        if(moteur.isRunning()) {
            return;
        }
        moteur.start();
        xFin = me.getX();
        yFin = me.getY();
    }//fin de la méthode MouseClicked

    public void mouseEntered(MouseEvent me) { }
    public void mouseExited(MouseEvent me) { }
    public void mousePressed(MouseEvent me) { }
    public void mouseReleased(MouseEvent me) { }

    @Override
    public void actionPerformed(ActionEvent ae) {
        if(ae.getSource() != moteur) {
            return;
        }
        i++;
        //if(x < xFin) {   FAUX
        if(i <= nb) {
            x = xIn + i*(xFin-xIn)/nb;
            y = yIn + i*(yFin-yIn)/nb;
        }else {
            xIn = xFin;
            yIn = yFin;
            i=0;
            moteur.stop();
        }
        repaint();
    }//fin de la méthode actionPerformed
}//fin de la classe interne Observer
```

1.4 On commence par quelques notions théoriques afin de pouvoir utiliser un "nouveau" contrôle swing, à savoir une case à cocher qui est un élément graphique atomique de type ***JCheckBox***.

La classe ***JCheckBox*** fait partie du package ***javax.swing*** et dispose, par exemple, d'un constructeur avec un seul argument de type ***String*** qui correspond au texte associé avec la case à cocher. Créée avec ce constructeur, la case à cocher n'est pas sélectionnée au début.

Afin de savoir :

- si une case à cocher est sélectionnée ou pas, on peut appeler la méthode d'instance sans argument ***isSelected( )*** de la classe ***JCheckBox*** qui retourne ***true*** si la case à cocher appelante est cochée ou ***false*** autrement ;
- quel est le texte associé à une case à cocher, on peut appeler la méthode d'instance sans argument ***getActionCommand( )*** de la classe ***JCheckBox*** qui retourne le texte associé à la case à cocher appelante comme ***String***.

Quand l'utilisateur clique avec la souris sur une case à cocher, elle change d'état :

- si elle n'était pas sélectionnée, elle devient sélectionnée (et une coche apparait dans la case) ;
- si elle était sélectionnée, elle devient non sélectionnée (et la case n'est plus cochée) ;
- dans les deux cas ci-dessus, un événement de type ***ItemEvent*** est lancé.

La classe ***ItemEvent*** fait partir du package ***java.awt.event***.

Afin de gérer les clics sur une case à cocher, il faut lui associer un écouteur de type ***ItemListener***.

L'interface ***ItemListener*** fait partie du package ***java.awt.event*** et prévoit une seule méthode d'instance avec l'en-tête suivant :

public void itemStateChanged (ItemEvent ie)

Il faut maintenant définir la classe interne ***PanCheck*** qui est privée et correspond à un container intermédiaire qui sera affiché dans la région inférieure du container de premier niveau. Cette classe écoute les événements produits par deux cases à cocher qu'elle contient.

Ecrivez ci-dessous la définition de la classe interne ***PanCheck*** (en suivant les consignes ci-dessus ainsi que celles données sous forme de commentaires).


```
//En-tête complet de la classe PanCheck
private class PanCheck extends JPanel
                        implements ItemListener
{
    //Déclaration d'un champ privé de type case à cocher nommé jcb1
    //(sans valeur initiale explicite)
    private JCheckBox jcb1;

    //Déclaration d'un champ privé de type case à cocher nommé jcb2
    //(sans valeur initiale explicite)
    private JCheckBox jcb2;

    //En-tête du constructeur sans argument et sans modificateur d'accès
    PanCheck()
    {
        //on met la couleur de fond du container à jaune
        //("yellow" en anglais)
        setBackground(Color.YELLOW);

        //on crée un objet de type case à cocher avec le texte associé
        //Cercle et on stocke son adresse dans le champ jcb1
        jcb1 = new JCheckBox("Cercle");

        //on ajoute le container qu'on est en train de créer comme
        //écouteur de la case à cocher référencée par jcb1
        jcb1.addItemListener(this);

        //on crée un objet de type case à cocher avec le texte associé
        //Carré et on stocke son adresse dans le champ jcb2
        jcb2 = new JCheckBox("Carré");
```

```
        //on ajoute le container qu'on est en train de créer comme
        //écouteur de la case à cocher référencée par jcb2
        jcb2.addItemListener(this);

        //on ajoute la case à cocher référencée par jcb1 au container
        add(jcb1);

        //on ajoute la case à cocher référencée par jcb2 au container
        add(jcb2);
    }//fin du constructeur panCheck

    @Override
    //En-tête du gestionnaire des événements produits par
    //les cases à cocher
    public void itemStateChanged(ItemEvent ie)
    {
        //on crée une variable locale de type case à cocher nommé
        //source et on y stocke la source de l'événement traité
        JCheckBox source = (JCheckBox)ie.getSource();

        //si la source de l'événement traité est la case à cocher
        //référencée par jcb1
        if(source == jcb1)
        {
            //si la case à cocher référencée par jcb1 est cochée
            if(jcb1.isSelected())
            {
```



```
                //stocker dans le champ forme le texte associé à la
                //case à cocher jcb1
                forme = jcb1.getActionCommand();
            }
            //autrement, si la case à cocher référencée par jcb2
            //est cochée
            else if(jcb2.isSelected())
            {
                //stocker dans le champ forme le texte associé à la
                //case à coche jcb2
                forme = jcb2.getActionCommand();
            }
        }
        //autrement, si la source de l'événement traité est la case à
        //cocher référencée par jcb2
        else if(source == jcb2)
        {
            //il n'y a pas besoin d'écrire cette partie
        }
        //on demande au champ surf de se redessiner
        surf.repaint();

    }//fin de la méthode gestionnaire d'événements
}//fin de la classe interne PanCheck
```

Question 2

Le but de cet exercice est d'écrire une classe nommé ***Rectangle*** qui nous permet de créer et de manipuler des objets correspondant à des rectangles. Chaque tel rectangle est caractérisé par deux paires de renseignements : les deux coordonnées de son coin supérieur gauche (à savoir l'abscisse et l'ordonnée de ce point) et ses deux dimensions (à savoir la largeur et la hauteur du rectangle).

On dispose d'une classe générique nommée ***PaireNum*** et dont le code est donné ci-dessous.

```
package cms_ctr4;

public class PaireNum<T extends Number> {
	private T premier;
	private T second;

	public T getPremier() {
		return premier;
	}
	public void setPremier(T premier) {
		this.premier = premier;
	}
	public T getSecond() {
		return second;
	}
	public void setSecond(T second) {
		this.second = second;
	}
	public PaireNum( ) { }
	public PaireNum(T premier, T second) {
		setPremier(premier);
		setSecond(second);
	}
	public PaireNum(PaireNum<T> source) {
		setPremier(source.premier);
		setSecond(source.second);
	}
	@Override
	public String toString() {
		return "(" + getPremier() + ", " + getSecond() + ")";
	}
}
```


Définir la classe ordinaire ***Rectangle*** qui :

- fait partie du même package que la classe ***PaireNum*** ;
- dérive (par héritage) d'une instanciation de la classe générique ***PaireNum*** pour la spécification de la variable de type par la classe enveloppe ***Integer*** => les champs hérités correspondent, dans cet ordre, à l'abscisse et à l'ordonnée du coin supérieur gauche du rectangle ;
- définit un champ propre nommé ***dims*** de type instanciation de la classe générique ***PaireNum*** pour la spécification de la variable de type par la classe enveloppe ***Integer*** => les deux éléments de ce champ correspondent, dans cet ordre, à la largeur et la hauteur du rectangle ;
- définit des méthodes d'altérations (une méthode ***getter*** et une méthode ***setter*** aussi simples que possible) pour le champ propre ***dims*** ;
- définit un constructeur public sans argument avec le corps vide (et qui ne fait rien) ;
- définit un constructeur public avec 4 arguments de type numérique primitif entier nommés ***x***, ***y***, ***w*** et ***h*** qui travaille ainsi :
  - la valeur du premier argument est encapsulée dans le premier champ hérité ;
  - la valeur du deuxième argument est encapsulée dans le deuxième champ hérité;
  - avec les deux derniers arguments on crée un nouvel objet du même type que le champ propre ***dims*** et on stocke sa référence dans le champ propre ***dims*** (par un appel au setter approprié) ;
- définit un constructeur par recopie public qui a un seul argument de type ***Rectangle*** et qui crée un nouvel objet qui est (en fait) un clone de l'objet argument ;
- définit une méthode publique qui peut être appelée avec le nom de la classe, nommée ***creerClone***, avec un seul argument de type ***Rectangle*** nommé ***orig*** et qui retourne comme valeur l'adresse d'un nouvel objet qui est un clone de l'objet argument ;
- une méthode d'instance publique nommé ***donnerCoteOppose*** sans argument et qui retourne un résultat de type classe ordinaire obtenue par l'instanciation de la classe générique ***PaireNum*** pour la spécification de la variable de type par la classe enveloppe ***Integer*** ; plus précisément, les éléments de la paire retournée par cette méthode doivent correspondre aux coordonnées du coin opposé au coin supérieur gauche (c'est-à-dire aux coordonnées du coin inférieur droit) de l'objet appelant ;
- définit une méthode d'instance publique nommée ***calculerAire*** sans argument et qui retourne une valeur de type numérique primitif entier qui correspond à l'aire du rectangle représenté par l'objet appelant.

Indiquer ci-dessous le code complet de la classe ***Rectangle***.


```
package cms_ctr4;

public class Rectangle extends PaireNum<Integer> {

	private PaireNum<Integer> dims;

	public PaireNum<Integer> getDims() {
		return dims;
	}

	public void setDims(PaireNum<Integer> dims) {
		this.dims = dims;
	}

	public Rectangle() { }

	public Rectangle(int x, int y, int w, int h) {
		setPremier(x);  //premier = x;  FAUX
		setSecond(y);   //second = y;   FAUX
		//dims = new PaireNum<>(w, h);  JUSTE mais pas de setter
		setDims(new PaireNum<>(w, h));
	}

	public Rectangle(Rectangle r) {
		//premier = r.premier;     FAUX (2 fautes)
		setPremier(r.getPremier());
		//second = r.second;       FAUX (2 fautes)
		setSecond(r.getSecond());
	//dims.setPremier(r.dims.getPremier());//NullPointerException
	//dims.setSecond(r.dims.getSecond());  //NullPointerException
		setDims(new PaireNum<>(r.dims.getPremier(),
		                       r.dims.getSecond()));
	}

	public static Rectangle creerClone(Rectangle orig) {
		return new Rectangle(orig);
	}

	public PaireNum<Integer> donnerCoteOppose() {
		return new PaireNum<>(getPremier() + dims.getPremier(),
		                      getSecond() + dims.getSecond());
	}

	public int calculerAire() {
		return dims.getPremier() * dims.getSecond();
	}
}
```

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

A part ce texte, cette page doit rester normalement blanche (mais vous pouvez l'utiliser pour la rédaction de vos réponses si la place prévue à cet effet s'est avérée insuffisante).