Extensions

Lucie YE

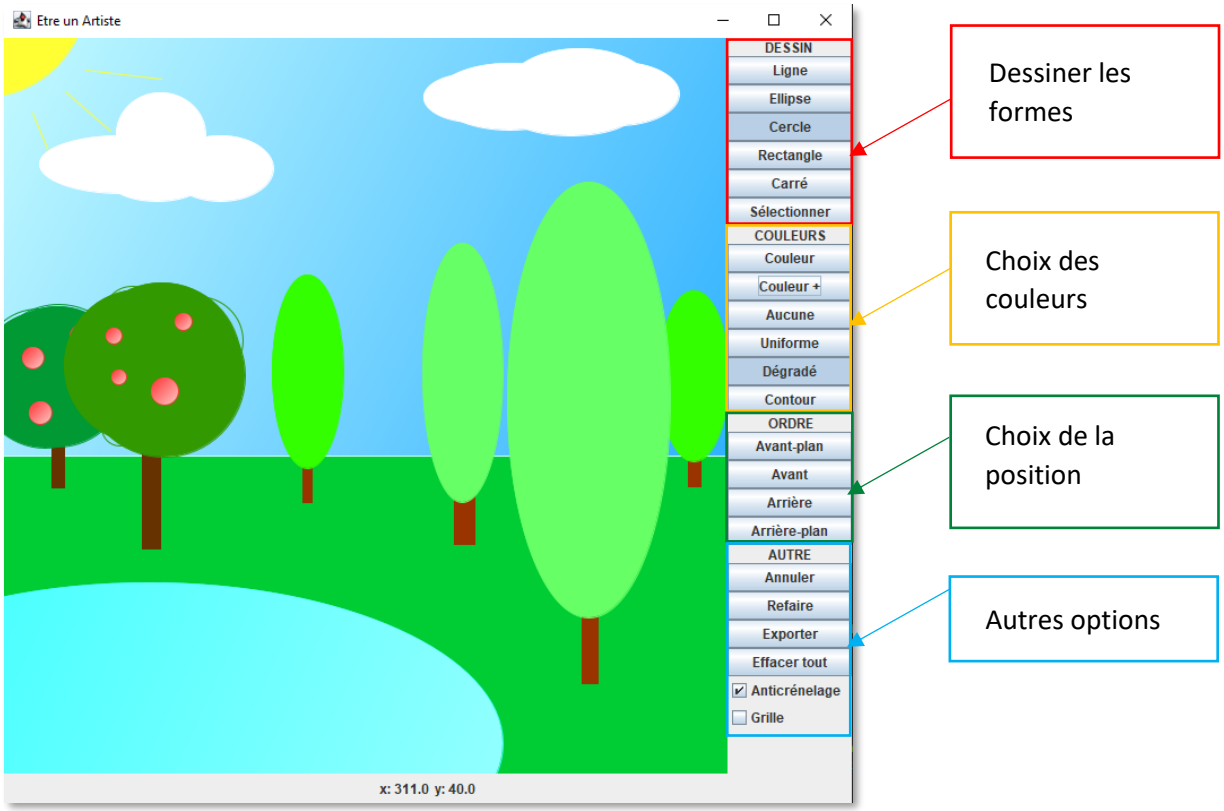
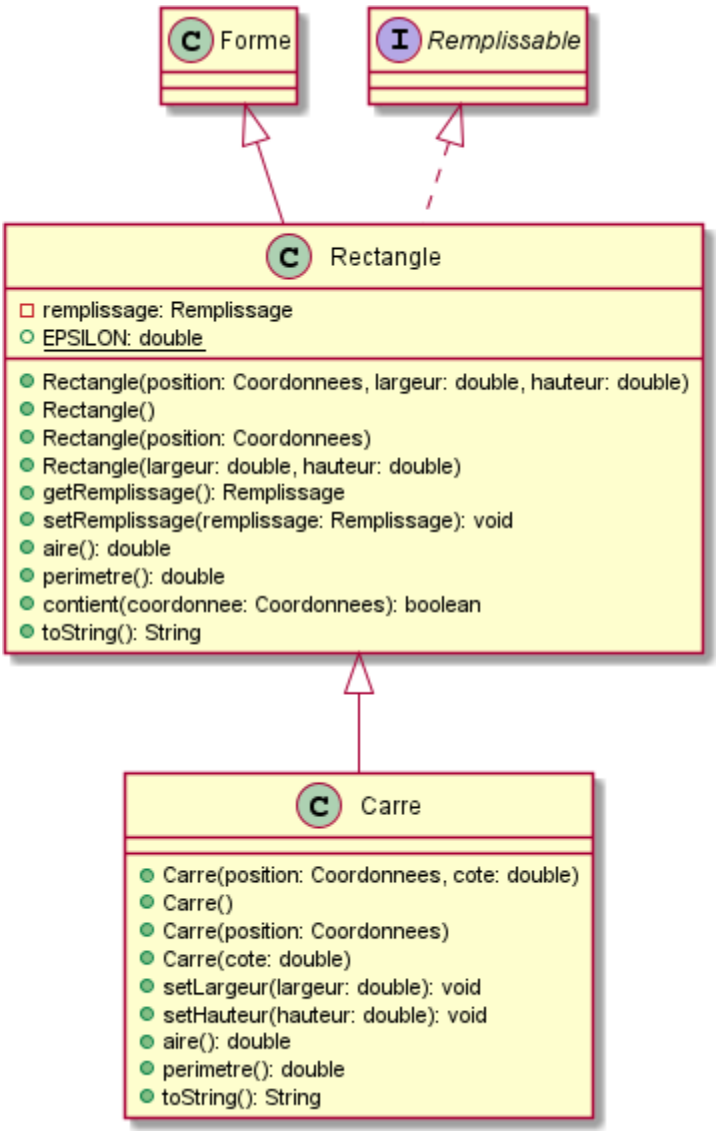


Table des matières

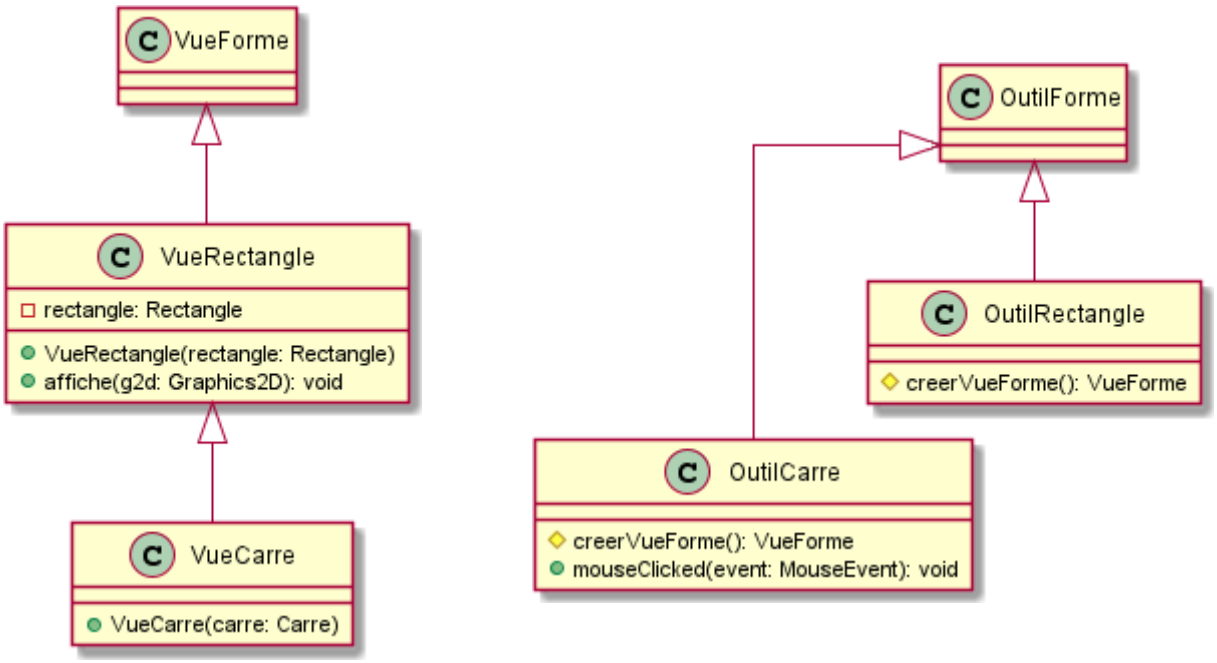
1. Dessiner des rectangles et des carrés

Cette extension permet de dessiner les rectangles et les carrés.

La classe pour dessiner un rectangle ressemble à celles des autres formes déjà présentes. Elle hérite la classe abstraite **Forme.java** et implémente l'interface **Remplissable.java**. Celle pour dessiner un carré hérite de la classe **Rectangle.java**, redéfinissant certaines méthodes spécifiques au carré.



De la même manière que les autres formes, il faut une classe **VueRectangle.java**, **VueCarre.java**, **OutilRectangle.java** et une classe **OutilCarre.java** pour permettre au *panneauDessin* d'associer le bon outil et de dessiner soit un rectangle, soit un carré.

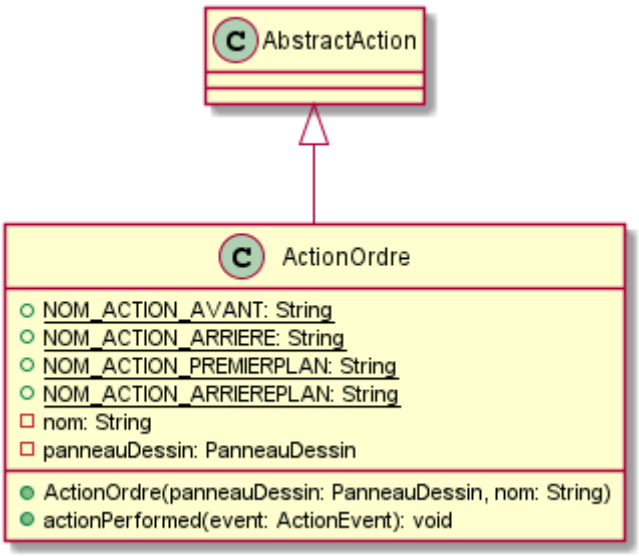


De plus, des nouvelles actions ont été ajoutés aux classes **ActionChoisirForme.java** pour permettre l'association de l'**OutilRectangle.java** et l'**OutilCarre.java**. Les boutons correspondant ont aussi été ajoutés dans **PanneauBarreOutil.java**.

2. Changer l'ordre

Cette extension permet de changer la position d'une forme par rapport à celles des autres.

La classe **ActionOrdre.java** est créée pour gérer les différents changements de position. En fonction du bouton appuyé, la machine à état change l'ordre de la liste **getVueFormes()** du *panneauDessin* grâce à des manipulations de liste.



```java
public void actionPerformed(ActionEvent event) {
    if(this.panneauDessin.getSelected() != null
            && this.panneauDessin.getVueFormes().size() > 0) {
        int i = this.panneauDessin.getVueFormes().indexOf(
                this.panneauDessin.getSelected());

        switch(this.nom) {
        case NOM_ACTION_ARRIERE:
            this.panneauDessin.getVueFormes().add(i-1,
                    this.panneauDessin.getSelected());
            this.panneauDessin.getVueFormes().remove(i+1);
            break;
        case NOM_ACTION_AVANT:
            this.panneauDessin.getVueFormes().add(i+2,
                    this.panneauDessin.getSelected());
            this.panneauDessin.getVueFormes().remove(i);
            break;
        case NOM_ACTION_ARRIEREPLAN:
            this.panneauDessin.getVueFormes().add(0,
                    this.panneauDessin.getSelected());
            this.panneauDessin.getVueFormes().remove(i+1);
            break;
        case NOM_ACTION_PREMIERPLAN:
            this.panneauDessin.getVueFormes().add(
                    this.panneauDessin.getVueFormes().size(),
                    this.panneauDessin.getSelected());
            break;
        default:
        }

        this.panneauDessin.repaint();
    }
}
```

3. Bouger une forme

Cette extension permet de bouger une forme lorsque celle-ci est sélectionnée.

La fonctionnalité est gérée par **OutilSelectionner.java**, donc une redéfinition de **mouseDragged()** a été effectuée. Ainsi lorsque l'utilisateur a sélectionné une forme et a appuyé la souris sur une coordonnée dans la forme, il peut glisser et changer la position de la forme.

```java
@Override
public void mouseDragged(MouseEvent event) {
    setFin(new Coordonnees(event.getX(), event.getY()));
    if(this.getPanneauDessin().getSelected() != null
            && this.getPanneauDessin().getSelected().getForme().contient(getDebut())) {
        double deltaX = getFin().getAbscisse() - getDebut().getAbscisse();
        double deltaY = getFin().getOrdonnee() - getDebut().getOrdonnee();

        this.formeSelectionnee.getForme().deplacerDe(deltaX, deltaY);
        setDebut(getFin());
        this.getPanneauDessin().repaint();
    }
}
```

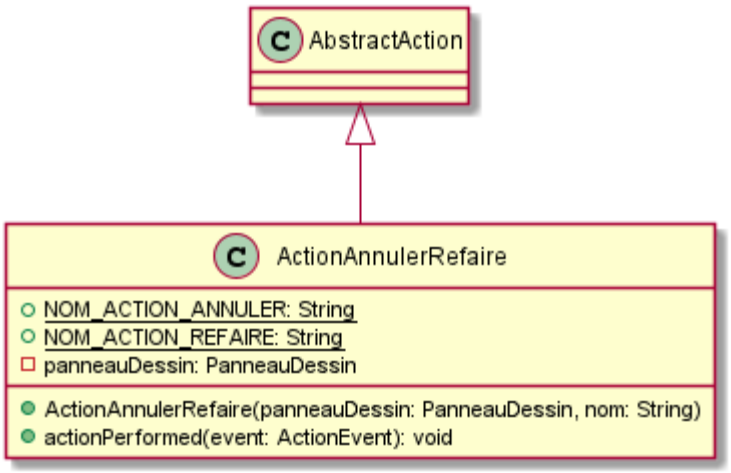
4. Effacer / Refaire une forme

Cette extension permet d'effacer ou de refaire une forme. Si une forme est sélectionnée, elle peut être effacée de cette manière.

Tout d'abord, une liste de *VueForme* a été créée dans la classe **PanneauDessin.java** pour enregistrer les formes à retirer / à refaire sur le *panneauDessin* :

```
private final List<VueForme> annulerRefaire;
```

La classe **ActionAnnulerRefaire.java** est ensuite créée pour manipuler les listes **getVueFormes()** et **getAnnulerRefaire()**.




```
public void actionPerformed(ActionEvent event) {
    if(event.getActionCommand().equals(NOM_ACTION_ANNULER)) {
        if(this.panneauDessin.getVueFormes().size() > 0) {
            VueForme aAnnuler = this.panneauDessin.getVueFormes().get(
                    this.panneauDessin.getVueFormes().size() - 1);
            if(this.panneauDessin.getSelected() != null) {
                aAnnuler = this.panneauDessin.getSelected();
            }
            this.panneauDessin.getAnnulerRefaire().add(aAnnuler);
            this.panneauDessin.getVueFormes().remove(aAnnuler);
            this.panneauDessin.setSelected(null);
        }
    }
    else if(event.getActionCommand().equals(NOM_ACTION_REFAIRE)) {
        if(this.panneauDessin.getAnnulerRefaire().size() > 0) {
            VueForme aRefaire = this.panneauDessin.getAnnulerRefaire().get(
                    this.panneauDessin.getAnnulerRefaire().size() - 1);
            this.panneauDessin.getVueFormes().add(aRefaire);
            this.panneauDessin.getAnnulerRefaire().remove(aRefaire);
        }
    }
    this.panneauDessin.repaint();
}
```

5. Modifier la couleur

Cette extension permet de changer la couleur d'une forme lorsqu'elle est sélectionnée.

La classe **ActionChoisirCouleur.java** est modifiée pour modifier la couleur de la forme sélectionnée, si une forme est bien sélectionnée. Après avoir mis à jour le *panneauDessin*, l'utilisateur peut continuer à dessiner.

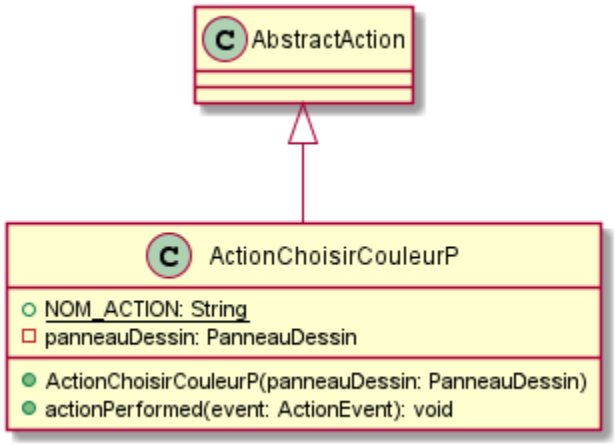
```
public void actionPerformed(ActionEvent event) {
    Color couleur = JColorChooser.showDialog(this.panneauDessin,
            NOM_ACTION, this.panneauDessin.getCouleurCourante());

    if(couleur != null) {
        if(this.panneauDessin.getSelected() != null) {
            this.panneauDessin.getSelected().getForme().setCouleur(couleur);
            this.panneauDessin.repaint();
        }
        this.panneauDessin.setCouleurCourante(couleur);
    }
}
```

6. Modes de remplissage : Dégradé et Contour

Cette extension permet d'avoir deux modes de remplissage en plus :

- *Dégradé : réalise un dégradé commençant avec la couleur choisit avec le bouton « Couleur » et finit le dégradé avec la couleur choisit avec le bouton « Couleur + »*
- *Contour : dessine une forme dont le contour est défini avec le bouton « Couleur » et le remplissage est défini par le bouton « Couleur + »*

Après avoir nommé les constantes « Dégradé » et « Contour » dans l'énumération **Remplissage.java**, il a fallu créer une autre variable de couleur de la même manière que la variable de couleur dans les fonctionnalités de base. Ensuite, pour passer les tests ACSM, une classe **ActionChoisirCouleurP.java** a été créée pour gérer cette nouvelle couleur.



Ces modes ne fonctionnent que pour les formes elliptiques, circulaires, rectangulaires et carrées.

MODE DÉGRADÉ

Dans les classes **VueEllipse.java** et **VueRectangle.java**, une section par rapport au dégradé a été ajoutée. L'instance **GradientPaint** ne s'occupant pas du remplissage de couleur, la forme à mettre en dégradé doit passer par le remplissage uniforme pour pouvoir s'afficher correctement. Un booléen est créé pour restaurer le mode de remplissage dégradé à la fin de la méthode.

Voici ce qui a été fait dans la méthode **affiche()** de **VueEllipse.java**. Le même code est écrit pour la méthode **affiche()** de **VueRectangle.java**.

```
g2d.setPaint(null);

boolean degrade = false;
if(this.ellipse.getRemplissage() == Remplissage.DEGRADE) {
    degrade = true;
    Color startColor = ellipse.getCouleur();
    Color endColor = getForme().getCouleurP();

    GradientPaint gradient = new GradientPaint(
            (int) ellipse.getCadreMinX(),
            (int) ellipse.getCadreMinY(),
            startColor,
            (int) ellipse.getCadreMaxX(),
            (int) ellipse.getCadreMaxY(),
            endColor);

    g2d.setPaint(gradient);
    this.ellipse.setRemplissage(Remplissage.UNIFORME);
}
```

MODE CONTOUR

Dans les classes **VueEllipse.java** et **VueRectangle.java**, une section par rapport au contour a aussi été ajoutée. Après avoir géré l'épaisseur du trait du contour, la forme sans remplissage est d'abord dessinée. Puis, la forme sans remplissage est d'abord dessinée, puis la forme passe par le mode remplissage uniforme pour son remplissage. Un booléen est créé pour restaurer le mode de remplissage dégradé à la fin de la méthode.

Voici ce qui a été fait dans la méthode **affiche()** de **VueEllipse.java**. Le même code est écrit pour la méthode **affiche()** de **VueRectangle.java**.

```
boolean contour = false;
if(this.ellipse.getRemplissage() == Remplissage.CONTOUR) {
    contour = true;
    Stroke ancien = g2d.getStroke();

    final BasicStroke plain = new BasicStroke(
            10f,
            BasicStroke.CAP_BUTT,
            BasicStroke.JOIN_MITER);

    g2d.setStroke(plain);
    g2d.drawOval((int) getForme().getPosition().getAbscisse(),
            (int) getForme().getPosition().getOrdonnee(),
            (int) getForme().getLargeur(), (int) getForme().getHauteur());

    g2d.setStroke(ancien);
    g2d.setColor(getForme().getCouleurP());
    this.ellipse.setRemplissage(Remplissage.UNIFORME);
}
```

7. Rubber banding

Cette extension permet de visualiser la forme lorsqu'utilisateur la dessine.

Une première modification a été portée dans la classe **PanneauDessin.java**. Une liste de *VueForme* a été créée pour enregistrer les formes à afficher au fur et à mesure que l'utilisateur glisse la souris :

```
private final List<VueForme> tracage;
```

Après avoir mis en place les accesseurs et la méthode pour ajouter des éléments dans la liste, la *VueForme* contenue dans la liste est dessinée sur le *panneauDessin* puis supprimée directement.

```
public void rubberbanding(Graphics2D g2D) {
    for (int j = 0 ; j<tracage.size(); j++) {
        tracage.get(j).affiche(g2D);
        tracage.remove(j);
    }
}
```

L'autre modification est effectuée dans **OutilForme.java**. Lorsque l'utilisateur glisse la souris. **mouseDragged()** place une coordonnée de fin puis ajoute la forme intermédiaire dans la liste à dessiner.

```
@Override
public void mouseDragged(MouseEvent event) {
    setFin(new Coordonnees(event.getX(), event.getY()));

    getPanneauDessin().ajouterTracage(creerVueForme());
    getPanneauDessin().repaint();
}
```

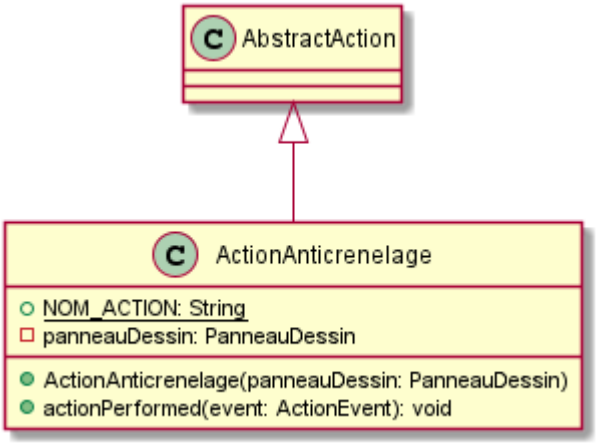
8. Anticrénelage

Cette extension permet d'activer l'anticrénelage lorsque l'utilisateur appuie sur le bouton correspondant.

Lorsque l'anticrénelage est activé, **paintComponent()** de **PanneauDessin.java** effectue la méthode **anticrenelage()**. Sinon, elle l'ignore.

```
public void anticrenelage(Graphics2D g2D) {
    RenderingHints rh = new RenderingHints(
            RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_OFF);

    if(getAnticrenelage()) {
        rh = new RenderingHints(
                RenderingHints.KEY_ANTIALIASING,
                RenderingHints.VALUE_ANTIALIAS_ON);
    }

    g2D.setRenderingHints(rh);
}
```

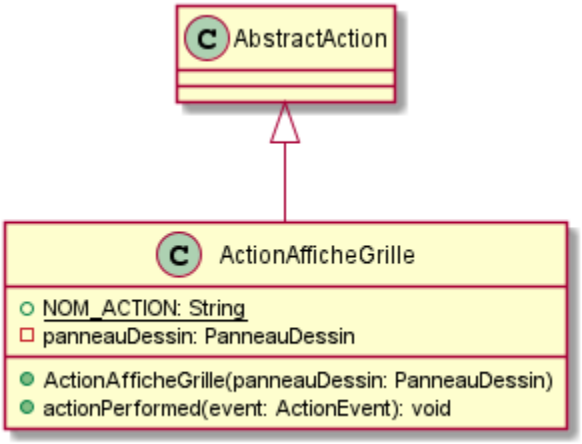
Pour ce faire, la classe **ActionAnticrenelage.java** est créée pour gérer l'activation et la désactivation de l'option anticrénelage du *panneauDessin*. Elle peut ensuite être invoquée par le bouton du *panneauBarreOutil* correspondant.



9. Affichage de la grille

Cette extension permet d'activer l'affichage d'une grille lorsque l'utilisateur appuie sur le bouton correspondant.

La classe **ActionAfficheGrille.java** est créée pour gérer l'activation et la désactivation de l'option sur le *panneauDessin*. Elle est ensuite invoquée par le bouton du *panneauBarreOutil* correspondant.



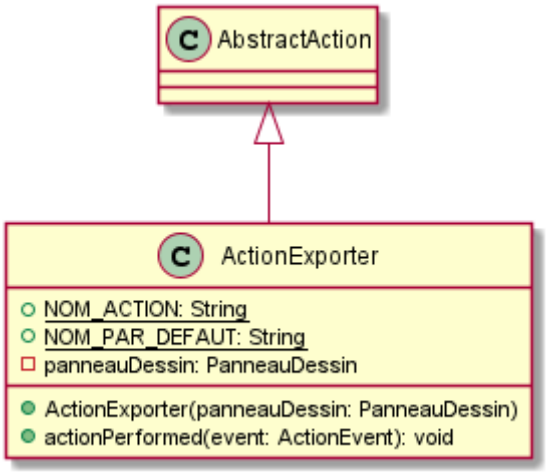
Puis, lorsque *panneauDessin* dessine les formes, l'utilisateur peut activer ou désactiver une grille en pointillé de couleur grise en arrière-plan.

```java
public void grille(Graphics2D g2D) {
    final float points[] = {2.0f};
    final BasicStroke dashed = new BasicStroke(
            0.25f,
            BasicStroke.CAP_ROUND,
            BasicStroke.JOIN_ROUND,
            5.0f, points, 5.0f);

    if(getGrille()) {
        Stroke ancien = g2D.getStroke();
        Color avant = g2D.getColor();
        g2D.setColor(new Color(230,230,230));
        g2D.setStroke(dashed);

        int cote = 20;
        int ligne = getHeight() / cote;
        int x = cote;
        for (int i = 0; i < ligne; i++) {
            g2D.drawLine(0, x, getWidth(), x);
            x = x + cote;
        }

        int colonne = getWidth() / cote;
        int y = cote;
        for (int i = 0; i < colonne; i++) {
            g2D.drawLine(y, 0, y, getHeight());
            y = y + cote;
        }

        g2D.setColor(avant);
        g2D.setStroke(ancien);
    }
}
```

10. Exportation de dessin

Cette extension permet d'exporter le dessin sous format 'jpg'.

La classe **ActionExporter.java** est créée pour exporter l'image sous format 'jpg' dans le fichier source, avec un nom choisi par l'utilisateur.



En créant une **BufferedImage** reliée au *panneauDessin*, qui hérite de **JPanel**, le dessin fait sur ce *panneauDessin* peut être enregistrée comme une **BufferedImage** et être exporté comme fichier image. Cette extension peut être invoquée en validant la fenêtre apparaissant après l'appui sur le bouton du *panneauBarreOutil* correspondant.

```
public void actionPerformed(ActionEvent event) {
    int action = JOptionPane.showConfirmDialog(
            this.panneauDessin,
            "Exporter?",
            NOM_ACTION,
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE);
    if(action == 0) {
        String nom = (String) JOptionPane.showInputDialog(
                this.panneauDessin,
                "Choisissez un nom",
                "Nom du nouveau dessin",
                JOptionPane.QUESTION_MESSAGE,
                null, null, null);
        if(nom.equals("")) {
            nom = NOM_PAR_DEFAUT;
        }

        BufferedImage bufferedImage = new BufferedImage(
                this.panneauDessin.getWidth(),
                this.panneauDessin.getHeight(),
                BufferedImage.TYPE_INT_RGB);

        Graphics2D g2d = bufferedImage.createGraphics();
        this.panneauDessin.paint(g2d);
        g2d.dispose();

        try {
            ImageIO.write(bufferedImage, "jpg", new File(nom + ".jpg"));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
  }
```