T.P. N° 1

maPage.htm

```
1  <HTML>
2  <HEAD>
3  <TITLE>Test de l'applet AppPaint</TITLE>
2  </HEAD>

4  <BODY>
3  <H1>Test de l'applet JAVA AppPaint</H1>
   <P>Cette page permet de tester l'applet JAVA <B>AppPaint</B>.
   Cette applet permet de faire un dessin sur l'espace occupé par l'applet
   et ceci en choisissant la couleur.
   </P>

5  <applet
        code="AppPaint.class"
        name="AppPaint"
        width=600
        height=400>
   </applet>

4  </BODY>
1  </HTML>
```

Commentaires

1 Le contenu d'une page HTML doit se placer en totalité entre les deux balises **<HTML>** et **</HTML>**. Ce qui se trouve en dehors de ces deux balises n'est pas interprété par les navigateurs.

2 Une page HTML est composée de deux parties. La première, qui constitue l'en-tête du document, est délimitée par les balises **<HEAD>** et **</HEAD>**. Son contenu n'est pas affiché par les navigateurs, mais est interprété.

3 Ce qui se trouve entre les balises **<TITLE>** et **</TITLE>** constitue le titre du document. Ce titre est affiché dans la barre de titre de la fenêtre (*caption*) du navigateur. Il ne faut pas confondre la balise **<TITLE>** et la balise **<H1>**. Cette dernière affiche un titre DANS le document et se trouve par conséquent, dans la seconde partie de la page HTML.

4 Une page HTML est composée de deux parties. La seconde, qui constitue les corps du document, est délimitée par les balises **<BODY>** et **</BODY>**. Elle contient TOUT ce qui doit être affiché par le navigateur.

5 Une applet JAVA est instanciée par la balise **<APPLET>**. Cette balise possède un certain nombre de propriétés de balises :

- **code** permet de définir dans quel fichier (extension **.class**) se trouve le code exécutable de l'applet. Ce fichier se trouve sur le serveur. Il faut peut être ajouter au nom du fichier un chemin relatif ou une URL complète selon la localisation exacte du fichier sur le serveur.

- **name** permet d'identifier l'applet en tant qu'objet dans la page HTML. Ce nom peut être utilisé par des scripts.

- **width** et **height** permet de définir en pixels la taille de l'applet dans la page HTML.

appPaint.java

```
1  import java.applet.*;
2  import java.awt.*;

3  public class AppPaint extends Applet
   {
4       public void paint(Graphics g)
        {
5            String strMessage = "Bonjour tout le monde !";
             g.drawString(strMessage, 10, 20);
        }
   }
```

Commentaires

1 Une applet est créée en dérivant la classe **Applet**. Cette classe est définie dans le package **java.applet** qui doit donc être importé.

2 Une applet fonctionne dans un environnement graphique dont tout les objets (comme la classe **Graphics** utilisée ici) sont définis dans le package **java.awt** qui doit donc être importé.

3 Une applet est déclarée en dérivant la classe **Applet**. Son identifiant (ici **AppPaint**) doit être le même que celui du nom de fichier (**AppPaint.class**) passé en paramètre **code** de la balise **<APPLET>**.

4 Chaque fois que l'applet doit être dessinée, elle reçoit le message **paint**. Il faut donc écrire la méthode **paint** pour dessiner le contenu de l'applet. Le dessin se fait dans un contexte d'affichage **g** de classe **Graphics**. Ce contexte d'affichage est reçu comme paramètre du message **paint**.

5 Dans une applet on n'utilise pas la fonction **println** pour afficher du texte. On utilise les méthodes de la classe **Graphics** pour dessiner (**drawString** pour afficher une chaîne de caractères de classe **String**).

*Ces deux fichiers se trouvent dans le répertoire **JATP1**.*

T.P. N° 2

maPage.htm

```
<HTML>
<HEAD>
<TITLE>Test de l'applet AppPaint</TITLE>
</HEAD>
<BODY>
<H1>Test de l'applet JAVA AppPaint</H1>
<P>Cette page permet de tester l'applet JAVA <B>AppPaint</B>.
Cette applet permet de faire un dessin sur l'espace occupé par l'applet
et ceci en choisissant la couleur.
</P>

<APPLET>                                                          1
      code="AppPaint.class"
      name="appPaint"
      width=600
      height=400>
      <PARAM name="background" value="#44CC88">                   2
      <PARAM name="foreground" value="#CC1122">
      <PARAM name="font" value="TimesRoman">
      <PARAM name="sizeFont" value="25">
      <PARAM name="styleFont" value="bold,italic">
      <PARAM name="message" value="Bonjour tout le monde !">
</APPLET>                                                         1

</BODY>
</HTML>
```

Commentaires

1 Une applet est définie dans une page HTML par la balise **<APPLET>**. Les paramètres de l'applet défini par la balise **<PARAM>** doivent être positionnés entre les balises **<APPLET>** et**</APPLET>**.

2 Les paramètres de l'applet sont définis par la balise **<PARAM>**. Cette balise possède deux propriétés : **name** pour identifier le paramètre de l'applet et **value** pour désigner la valeur de ce paramètre. Pour cette applet, 6 paramètres sont définis :

- **background** pour la couleur d'arrière-plan. Ce paramètre doit être exprimé comme une chaîne de caractères au format standard des couleurs HTML, à savoir **"#RRVVBB"**.
- **foreground** pour la couleur d'affichage du message. Ce paramètre doit être exprimé comme une chaîne de caractères au format standard des couleurs HTML, à savoir **"#RRVVBB"**.
- **font** pour la police de caractère utilisée pour l'affichage du message. Les valeurs de ce paramètre ne peuvent être que **"Helvetica"**, **"TimesRoman"**, **"Courrier"**, **"Dialog"** ou **"ZapfDingbats"**.
- **sizeFont** pour le corps de la police utilisée pour l'affichage du message.

- **styleFont** pour le style de la police utilisée pour l'affichage du message. La valeur de ce paramètre est une chaîne de caractères contenant des mots clés éventuellement séparés par des virgules : **bold** pour gras et **italic** pour italique.

- **message** pour le texte du message à afficher.

appPaint.java

```
import java.awt.*;
import java.applet.*;

public class AppPaint extends Applet
{
     // Déclaration des valeurs par défaut des paramètres de l'applet
     // passés par la balise HTML <PARAM>
1    private final Color DEFAULT_background = new Color(255,255,255);
     private final Color DEFAULT_foreground = new Color(0,0,0);
     private final String DEFAULT_font = "Helvetica";
     private final int DEFAULT_sizeFont = 16;
     private final int DEFAULT_styleFont = 0;
     private final String DEFAULT_message = "Bonjour";

     // Déclaration des noms des paramètres de l'applet
     // passés par la propriété name de la balese HTML <PARAM>
2    private final String PARAM_background = "background";
     private final String PARAM_foreground = "foreground";
     private final String PARAM_font = "font";
     private final String PARAM_sizeFont = "sizeFont";
     private final String PARAM_styleFont = "styleFont";
     private final String PARAM_message = "message";

     // Déclaration des propriétés d'échange pour les paramètres de
     // l'applet passés par la balise HTML <PARAM>
3    private Color m_background = DEFAULT_background;
     private Color m_foreground = DEFAULT_foreground;
     private String m_font = DEFAULT_font;
     private int m_sizeFont = DEFAULT_sizeFont;
     private int m_styleFont = DEFAULT_styleFont;
     private String m_message = DEFAULT_message;

4    public void init()
     {
          usePageParams();
     }

     public void paint(Graphics g)
     {
5         Dimension di = getSize();
6         Font f = new Font(m_font, m_styleFont, m_sizeFont);
          g.setFont(f);
7         g.setColor(m_foreground);
8         FontMetrics fm = getFontMetrics(f);
          int nSizeString = fm.stringWidth(m_message);
          int x = (di.width - nSizeString) / 2;
          int y = di.height / 2;
9         g.drawString(m_message, x, y);
     }

     private void usePageParams()
     {
10        String strParam = getParameter(PARAM_background);
          if (strParam != null)
          {
11             try
               {
                    m_background = stringToColor(strParam);
```

```
                    }
                    catch (NumberFormatException e)
                    {
                    }
              }

10            strParam = getParameter(PARAM_foreground);
              if (strParam != null)
              {
11                  try
                    {
                        m_foreground = stringToColor(strParam);
                    }
                    catch (NumberFormatException e)
                    {
                    }
              }

10            strParam = getParameter(PARAM_font);
              if (strParam != null)
              {
                    m_font = strParam;
              }

10            strParam = getParameter(PARAM_sizeFont);
              if (strParam != null)
              {
12                  try
                    {
                        Integer intParam = new Integer(strParam);
                        m_sizeFont = intParam.intValue();
                    }
                    catch (NumberFormatException e)
                    {
                    }
              }

10            strParam = getParameter(PARAM_styleFont);
              if (strParam != null)
              {
13                  try
                    {
                        for (int i = 1; true; i++)
                        {
                            String strStyle = fieldOf(strParam, i , ',');
                            if (strStyle.compareTo("bold") == 0)
                            {
                                m_styleFont += Font.BOLD;
                            }
                            else if (strStyle.compareTo("italic") == 0)
                            {
                                m_styleFont += Font.ITALIC;
                            }
                        }
                    }
                    catch (IndexOutOfBoundsException e)
                    {
                    }
              }


10            strParam = getParameter(PARAM_message);
              if (strParam != null)
              {
                    m_message = strParam;
              }
14            this.setBackground(m_background);
              this.setForeground(m_foreground);
        }

15      public String[][] getParameterInfo()
        {
              String[][] info =
              {
                    { PARAM_background, "String",
```

```
                        "Couleur d'arrière-plan, format \"#rrggbb\"" },
                    { PARAM_foreground, "String",
                        "Couleur de premier plan, format \"#rrggbb\"" },
                    { PARAM_font, "String",
                        "Nom de la police de caractère" },
                    { PARAM_sizeFont, "Integer",
                        "Corps de la police de caractères" },
                    { PARAM_styleFont, "String",
                        "Style de la police de caractères" },
                    { PARAM_message, "String",
                        "Message à afficher dans l'applet" }

              };
              return info;
        }


16      private Color stringToColor(String paramValue)
              throws NumberFormatException
        {
              try
              {
17                int red =
              (Integer.decode("0x" + paramValue.substring(1,3))).intValue();
                  int green =
              (Integer.decode("0x" + paramValue.substring(3,5))).intValue();
                  int blue =
              (Integer.decode("0x" + paramValue.substring(5,7))).intValue();
                  return new Color(red,green,blue);
              }
18            catch (StringIndexOutOfBoundsException e)
              {
                  throw new NumberFormatException();
              }
        }


19      private String fieldOf(String paramValue, int nField, int sep)
              throws IndexOutOfBoundsException
        {
20            if (nField < 1) throw new IndexOutOfBoundsException();
              int nDebut = 0;
              String strResult = "";
              for (int i = 0; i < nField; i++)
              {
                  int nFin = paramValue.indexOf(sep, nDebut);
                  if (nFin == -1)
                  {
21                    strResult = paramValue.substring(nDebut);
                      nDebut = paramValue.length() + 1;
                  }
                  else
                  {
                      strResult = paramValue.substring(nDebut, nFin);
                      nDebut = nFin + 1;
                  }
              }
              return strResult;
        }

  }
```

Commentaires

1 Pour chaque propriété on définit une valeur par défaut. Cette valeur est déclarée en constante (mot-clé **final**). Son identifieur est préfixé **DEFAULT_**.

2 Pour chaque propriété, il est nécessaire de définir un nom. Ce nom est une chaîne de caractères dont le contenu doit être identique au paramètre de la

propriété **name** de balise **<PARAM>**. Le nom est déclaré en constante (mot-clé **final**) et son identifiant est préfixé **PARAM_**.

3 Pour chaque propriété, une donnée membre est déclarée pour la classe de l'applet. Ces données membres sont initialisée par leur valeur par défaut.

4 L'initialisation de ces données membres doit ce faire dans la méthode **init**. Cette méthode est appelée une fois, juste après le chargement de l'applet. Le rôle de la méthode **init** n'est pas seulement d'initialiser les données membres correspondant aux propriétés de l'applet. C'est pourquoi l'usage veut que l'on isole cette initialisation dans une autre méthode **usePageParam**, cette dernière étant appelée elle-même par **init**.

5 La méthode **paint** est appelée chaque fois que l'applet a besoin d'être redessinée. Comme le message doit être centré dans l'applet, il est nécessaire d'en connaître la taille. On utilise la méthode **getSize** de la classe **Applet** pour calculer **di** un objet de classe **Dimension**. Cette classe définit deux propriétés **width** et **height** correspondant respectivement à une largeur et une hauteur de rectangle.

6 Pour afficher le message avec la police de caractères dont les caractéristiques sont définies dans les propriétés **font**, **sizeFont** et **styleFont** de l'applet, il faut d'abord créer un objet de classe **Font** en le construisant sur la base de ces paramètres. La police ainsi crée est affecté au contexte d'affichage **g** par la méthode **setFont** de la classe **Graphics**.

7 Pour afficher le message avec la couleur définie dans la propriété **foreground** de l'applet, on utilise la méthode **setColor** de la classe **Graphics** pour modifier la couleur du contexte d'affichage **g**.

8 Pour centrer le texte dans l'applet, il faut calculer l'espace que va occuper ce texte. Cet espace dépend du type, du style et de la taille de la police utilisée. Pour cela, on calcule les paramètres numériques de la police utilisée, rassemblés dans **fm**, objet de la classe **FontMetrics**, en utilisant la méthode **getFontMetrics**. La méthode **stringWidth** de la classe **FontMetrics** permet de calculer l'espace **nSizeString** que va occuper le message **m_message** sur l'applet. Il est alors facile de calculer **x** et **y**, la position du premier caractère de **m_message** pour centrer ce dernier dans l'applet.

9 **m_message** est affiché en position **x**, **y** pour que le message soit centré sur l'applet.

10 La méthode **usePageParams** est appelée par la méthode **init**, pour initialiser les données membres relatives aux propriétés de l'applet définie grâce aux balises HTML **<PARAM>**. Pour chaque propriété, on va extraire la valeur (paramètre de la propriété **value** de la balise **<PARAM>**), en utilisant la méthode **getParameter** avec le nom de la propriété (paramètre de la propriété **name** de la balise **<PARAM>**). Le résultat obtenu est de type **String**. Si **getParameter** renvoi **null** comme résultat, c'est que l'utilisateur de l'applet a omis de définir la propriété de l'applet dans sa page HTML. Auquel cas, on conserve la valeur par défaut précédemment initialisée.

11 **getParameter** fourni un résultat de type **String**. Pour les propriétés de type **Color**, il est donc nécessaire de convertir la propriété en objet de classe **Color**. Pour cela, on utilise la méthode **stringToColor** décrite ci-après. Pour que la méthode **stringToColor** fonctionne correctement, il faut que la chaîne de caractères fournie soit au format standard des couleurs HTML, à savoir **"#RRVVBB"**. **RR**, **VV**, **BB** étant les valeurs hexadécimales sur deux chiffres correspondant respectivement aux composantes rouge, verte et bleue de la couleur. Si l'utilisateur de l'applet s'est trompé, **stringToColor** génère une exception de classe **NumberFormatException**. La donnée membre correspondante n'est pas modifiée et conserve sa valeur par défaut.

12 **getParameter** fourni un résultat de type **String**. Pour les propriétés de type **int**, il est donc nécessaire de convertir la propriété en un nombre de type **int**. Pour cela, on utilise la classe **Integer** qui construit un entier à partir d'une chaîne de caractères. Si l'utilisateur de l'applet a utilisé des caractères non numériques, **Integer** génère une exception de classe **NumberFormatException**. La donnée membre correspondante n'est pas modifiée et conserve sa valeur par défaut.

13 La propriété styleFont est une chaîne de caractère composée de plusieurs styles séparés par des virgules. Comme ni le nombre, ni l'ordre des styles utilisés n'es connu, on utilise une boucle sans fin pour explorer tous les champs de la chaîne. Pour cela, on utilise la méthode **fieldOf** décrite ci-après. Chaque style étant ainsi isolé, on le compare avec leurs valeurs possibles et on met à jour la donnée membre correspondante (**m_styleFont**). La boucle s'arrête quand **fieldOf** génère une exception de classe **IndexOutOfBoundException**. C'est à dire quand on cherche à extraire un champ qui n'existe pas.

14 Les couleurs d'arrière-plan et d'affichage sont modifiées en fonctions des propriétés paramétrées par les méthodes **setBackground** et **setForeground** de la classe **Applet**.

15 **getParameterInfo** est une méthode qui est appelée par les environnements de développement utilisés pour créer des pages HTML. Elle doit fournir en résultat un tableau de chaînes de caractères à 3 colonnes correspondant respectivement au nom des propriétés, à leur type et à une brève description.

16 **stringToColor** permet de convertir un paramètre de type chaîne de caractères au format **"#RRVVBB"** (**RR**, **VV**, **BB** étant les valeurs hexadécimales sur deux chiffres correspondant respectivement aux composantes rouge, verte et bleue de la couleur) en un objet de classe **Color**. Si la chaîne de caractères n'est pas conforme à ce format, une exception de classe **NumberFormatException** est générée.

17 Pour cela on extrait chaque sous chaîne de composante avec la méthode **substring** de la classe **String** et on la convertit en entier par la méthode **decode** de la classe **Integer**. Si des caractères non hexadécimaux sont utilisés une exception de classe **NumberFormatException** est générée.

18 Si l'utilisateur n'a pas fourni assez de caractères, **substring** va générer une exception de classe **StringIndexOutOfBoundsException**. Cette exception est capturée par **try** pour être transformée en une exception de classe **NumberFormatException** dans **catch**. De cette façon, si la chaîne n'est pas au format **"#RRVVBB"**, une exception de classe unique est générée. Exception qui peut être capturée par les méthodes utilisant **stringToColor**.

19 **fieldOf** permet d'extraire un le champ N° **nField** dans la chaîne **paramValue** dans laquelle les champs sont séparés par le caractère **sep** (en général une virgule). Si **nField** ne correspond à aucun champ de la chaîne, une exception de classe **IndexOutOfBoundsException** est générée.

20 Si le numéro de champ est inférieur à 1, une exception de classe **IndexOutOfBoundsException** est générée.

21 Si le numéro de champ est supérieur au nombre de champs contenus dans la chaîne, **substring** va générer une exception de classe **StringIndexOutOfBoundsException**. Cette classe étant elle-même dérivée de **IndexOutOfBoundsException**, il suffira de capturer les exceptions de cette dernière classe dans les méthodes appelant **fieldOf**. pour signaler que le numéro de champ **nField** ne correspond pas à un n° de champ valide.

*Ces deux fichiers se trouvent dans le répertoire **JATP2**.*

T.P. N° 3

maPage.htm

```
<HTML>
<HEAD>
<TITLE>Test de l'applet AppPaint</TITLE>
</HEAD>
<BODY>
<H1>Test de l'applet JAVA AppPaint</H1>
<P>Cette page permet de tester l'applet JAVA <B>AppPaint</B>. Cette
applet permet de faire un dessin sur
l'espace occup&eacute; par l'applet et ceci en choisissant la couleur.
</P>
<APPLET
     code="AppPaint.class"
     codeBase=" "
     name=appPaint
     height=400
     width=600 >
     <PARAM NAME="foreground" VALUE="#000000">
     <PARAM NAME="background" VALUE="#C0C0C0">
</APPLET>
</BODY>
</HTML>
```

appPaint.java

```
import java.awt.*;
import java.applet.*;
import java.util.*;

public class AppPaint extends Applet
{
     // Déclaration des données membres de l'applet
1    private Point m_lastPos = new Point(0, 0);
2    private Vector m_dessin = new Vector();

     // Déclaration des valeurs par défaut des paramètres de l'applet
     // passés par la balise HTML <PARAM>
     private final Color DEFAULT_background = new Color(255,255,255);
     private final Color DEFAULT_foreground = new Color(0,0,0);

     // Déclaration des noms des paramètres de l'applet
     // passés par la propriété name de la balese HTML <PARAM>
     private final String PARAM_background = "background";
     private final String PARAM_foreground = "foreground";

     // Déclaration des propriétés d'échange pour les paramètres de
     // l'applet passés par la balise HTML <PARAM>
     private  Color m_background = DEFAULT_background;
     private  Color m_foreground = DEFAULT_foreground;

     public void init()
     {
          usePageParams();
     }

     public void paint(Graphics g)
     {
3         Point lastPos = new Point(0,0);
          for (Enumeration e = m_dessin.elements(); e.hasMoreElements();)
          {
               Object o = e.nextElement();
4              if (o instanceof Color)
               {
                    Color c = (Color)o;
                    g.setColor(c);
```

5

```
                    lastPos = (Point)e.nextElement();
                }
                else if (o instanceof Point)
                {
                    Point pt =(Point)o;
                    g.drawLine(lastPos.x, lastPos.y, pt.x, pt.y);
                    lastPos = pt;
                }
            }
        }

        private void usePageParams()
        {
            /**
             * Lecture des paramètres de l'applet
             * <PARAM NAME="background" VALUE="#rrggbb">
             * <PARAM NAME="foreground" VALUE="#rrggbb">
             * à partir de l'hôte HTML de l'applet.
             */

            String strParam = getParameter(PARAM_background);
            if (strParam != null)
            {
                try
                {
                    m_background = stringToColor(strParam);
                }
                catch (NumberFormatException e)
                {
                }
            }

            strParam = getParameter(PARAM_foreground);
            if (strParam != null)
            {
                try
                {
                    m_foreground = stringToColor(strParam);
                }
                catch (NumberFormatException e)
                {
                }
            }
            this.setBackground(m_background);
            this.setForeground(m_foreground);
        }

        public String[][] getParameterInfo()
        {
            String[][] info =
            {
                { PARAM_background, "String",
                    "Couleur d'arrière-plan, format \"#rrggbb\"" },
                { PARAM_foreground, "String",
                    "Couleur de premier plan, format \"#rrggbb\"" },
            };
            return info;
        }

        private Color stringToColor(String paramValue)
            throws NumberFormatException
        {
            try
            {
                int red =
            (Integer.decode("0x" + paramValue.substring(1,3))).intValue();
                int green =
            (Integer.decode("0x" + paramValue.substring(3,5))).intValue();
                int blue =
            (Integer.decode("0x" + paramValue.substring(5,7))).intValue();
                return new Color(red,green,blue);
            }
            catch (StringIndexOutOfBoundsException e)
            {
                throw new NumberFormatException();
```

```
            }
        }


6       public boolean mouseDown(Event e, int x, int y)
        {
            m_dessin.addElement(m_foreground);
            Point pos = new Point(x, y);
            m_dessin.addElement(pos);
            m_lastPos = pos;
            return super.mouseDown(e, x, y);
        }

7       public boolean mouseDrag(Event e, int x, int y)
        {
            Point pos = new Point(x, y);
            m_dessin.addElement(pos);
8           Graphics g = getGraphics();
            g.setColor(m_foreground);
            g.drawLine(m_lastPos.x, m_lastPos.y, x, y);
            m_lastPos = pos;
            return super.mouseDrag(e, x, y);
        }

9       public boolean mouseUp(Event e, int x, int y)
        {
            Point pos = new Point(x, y);
            m_dessin.addElement(pos);
            repaint();
            return super.mouseUp(e, x, y);
        }

10      public boolean keyDown(Event e, int key)
        {
            switch (key)
            {
            case Event.DELETE:
                m_dessin.removeAllElements();
                repaint();
                break;
            }
            return super.keyDown(e, key);
        }
}
```

Commentaires

1 Comme les messages de la souris sont indépendants, il faut mémoriser dans une donnée membre (**m_lastPos**) la dernière position de la souris afin de pouvoir tracer, en réponse à chaque message, le trait entre la position précédente et la position du curseur de la souris reçue en paramètre du message.

2 Comme le message **paint** peut être reçu à tout moment par l'applet, il faut mémoriser quelque part tous les points qui composent le dessin. On prévoit donc une donnée membre supplémentaire (**m_dessin**) de type collection pour contenir tous ces points. La classe **Vector** a été choisie car l'ordre des points est important.

3 Lors du traitement de la méthode **paint**, il faut faire une itération de la collection **m_dessin** pour récupérer, un par un, tous les éléments qu'elle contient. Ces éléments sont de deux natures : de classe **Color** et de classe **Point**.

4 Si l'élément sorti de la collection est de classe **Color**, il s'agit d'un début de coup de crayon. Auquel cas on modifie la couleur de tracé par un appel à la méthode **setColor** de la classe **Graphic** et on cherche l'élément suivant qui est forcément un objet de classe **Point**. Ce point va servir d'origine au tracé du coup de crayon et est mémorisé dans la variable **lastPos**.

5 Si l'élément sorti de la collection est de classe **Point**, on trace le trait qui le relie à la position précédente (**lastPos**) et on actualise **lastPos** avec l position actuelle.

6 La méthode **mouseDown** est appelée lorsque l'utilisateur enfonce le bouton de la souris. Cela signifie qu'il s'agit du premier point d'un coup de crayon. On mémorise, dans la collection **m_dessin**, la couleur actuelle de tracé (**m_foreground**) ainsi que la position du curseur de la souris (**x**, **y**) en tant qu'objet de la classe **Point**. Le résultat renvoyé est le résultat calculé par la méthode de la classe parente.

7 La méthode **mouseDrag** est appelée lorsque la souris est déplacée, le bouton enfoncé. Il s'agit donc d'un point composant le coup de crayon en cours de tracé. Ce point est enregistré dans la collection **m_dessin**.

8 Pour afficher le dessin au fur et à mesure que l'utilisateur procède au tracé, il faut relier le point reçu avec le point précédent. Pour dessiner, il est indispensable de disposer du contexte d'affichage de l'applet. Ce contexte d'affichage est obtenu par la méthode **getGraphics**. Il est alors possible de relier le point paramètre du message et la position précédente du curseur de la souris.

9 La méthode **mouseUp** est appelée lorsque l'utilisateur relâche le bouton de la souris. Cela signifie qu'il s'agit du dernier point composant le coup de crayon. Ce point est bien sur enregistré dans la collection **m_dessin**. Puis un message **paint** est envoyé à l'applet, par la méthode **repaint**, pour qu'elle redessine proprement le dessin complet.

10 La méthode **keyDown** est appelée lorsque l'utilisateur tape une touche au clavier. Seule la touche **Suppr** est à traiter. On vide alors la collection de tous ses éléments (méthode **removeAllElements**) et on envoie un message **paint** à l'applet (méthode **repaint**) pour effacer physiquement le dessin.

*Ces deux fichiers se trouvent dans le répertoire **JATP3**.*

T.P. N° 4

maPage.htm

```
1
2
3
4
```

Commentaires

1 .

appPaint.java

```
1
2
3
4
5
6
```

Commentaires

3 1

*Ces deux fichiers se trouvent dans le répertoire **JATP4**.*