SOMMAIRE

SOMMAIRE....1

INTRODUCTION....3

T.P. N °1 - UNE APPLET SIMPLE....5

1.1 Objectifs....5

1.2 Ce qu'il faut savoir....5

1.2.1Une applet JAVA, qu'est-ce-que c'est ? 5

1.2.2 Structure d'une applet....5

1.2.3 Instanciation d'une applet dans une page HTML....5

1.2.4Affichage d'une applet....6

1.2.5Surcharge de la méthode paint....7

1.3 Travail à réaliser....8

T.P. N°2 - PARAMÈTRES D'UNE APPLET....9

2.1 Objectifs....9

2.2 Ce qu'il faut savoir....9

2.2.1 Paramétrage d'une applet en HTML....9

2.2.2Exploitation des paramètres dans l'applet....10

2.2.3Propriétés de l'applet associées aux paramètres....10

2.2.4Cycle de vie d'une applet....11

2.2.5Initialisation des propriétés de l'applet....12

2.2.6Surcharge de la méthode getParameterInfo....15

2.3 Travail à réaliser....17

T.P. N°3 - INTERACTIONS UTILISATEURS....19

3.1 Objectifs....19

3.2 Ce qu'il faut savoir....19

3.2.1Les interactions utilisateurs....19

3.2.2Les messages du clavier....19

3.2.3Les messages de la souris....21

3.2.4Traitement des messages de la souris....23

3.3 Travail à réaliser....24

CONCLUSION....25

INDEX ANALYTIQUE....27

INTRODUCTION

Cette troisième partie traite partie traite du développement d'***applets*** JAVA. Les ***applets*** sont des programmes JAVA insérés, en tant que composants actifs, dans des pages HTML. Les programmes réalisés dans ce support fonctionneront en mode graphique (dans la fenêtre d'un navigateur comme Explorer, par exemple).

Ce support de formation est constitué d'une liste d'exercices permettant de s'approprier les différentes difficultés du développement d'***applets*** en JAVA. Bien que les corrigés types de chaque étapes soient présenté dans des répertoires séparés, il s'agit souvent de la même applet qui évolue, étape par étape, jusqu'à sa version définitive, à savoir une applet qui permet de dessiner, de façon similaire aux applications de Windows, sur une page HTML.

Chaque exercice est structuré de la façon suivante :

- Description des objectifs visés.
- Explications des techniques à utiliser (Ce qu'il faut savoir).
- Enoncé du problème à résoudre (Travail à réaliser).
- Renvois bibliographiques éventuels dans les ouvrages traitant de ces techniques (Lectures).

Tous ces exercices sont corrigés et commentés dans le document intitulé :

- Proposition de corrigé JAVA_CPC
  Apprentissage d'un langage de Programmation Orientée Objet
  JAVA (applet).

Pour pouvoir utiliser ce support, les bases de la programmation en langage JAVA et les concepts de la programmation orientée objet doivent être acquis. Ces techniques sont traitées dans les documents suivants :

- Support de formation JAVA_ASF
  Apprentissage du langage de programmation JAVA (bases)

- Support de formation JAVA_ASF
  Apprentissage du langage de programmation JAVA (objet)

Par ailleurs, es ***applets*** sont des programmes JAVA insérés, en tant que composants actifs, dans des pages HTML. Le langage HTML doit donc être également maîtrisé.

Les ouvrages auxquels il sera fait référence dans le présent support sont les suivants :

- **JAVA (Le Macmillan)**
  Alexander Newman
  Editions Simon & Schuster Macmillan

- **Formation à JAVA**
  Stephen R. Davis
  Editions Microsoft Press

- **JAVA (Megapoche)**
  Patrick Longuet
  Editions Sybex

T.P. N °1 - UNE APPLET SIMPLE

1.1 OBJECTIFS

- Déclaration d'une ***applet***
- La classe **`Applet`**
- La méthode **`paint`**
- Comment insérer une ***applet*** dans une page HTML.

1.2 CE QU'IL FAUT SAVOIR

1.2.1 Une applet JAVA, qu'est-ce-que c'est ?

Une ***applet*** est un programme JAVA qui peut être inséré comme composant actif d'une page HTML. C'est donc une application graphique qui ne peut fonctionner que dans un navigateur comme **Explorer** (Microsoft) ou **Navigator** (NetScape) pour ne citer que les plus célèbres.

1.2.2 Structure d'une applet

Une **applet** est déclarée comme une classe JAVA en dérivant la classe de base **`Applet`** (et non plus **`Object`** comme on l'a fait jusqu'à présent). **`Applet`** est elle-même une classe dérivée de **`Object`**.

```
public class MonApplet extends Applet
{
    // Déclaration des méthode de l'applet
}
```

`MonApplet` est l'identifiant de la nouvelle classe de l'***applet*** que l'on est en train de développer.

La classe **`Applet`** est déclarée dans le ***package*** **`java.applet`**. Ce ***package*** doit être importé.

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

1.2.3 Instanciation d'une applet dans une page HTML

Une ***applet*** JAVA ne peut être instanciée que dans une page HTML. On utilise pour cela la balise **`<APPLET>`**.

```
<HTML>
<HEAD>
<TITLE>Test de l'applet MonApplet</TITLE>
</HEAD>
<BODY>
<H1>Test de l'applet JAVA MonApplet </H1>
<P>Cette page permet de tester l'applet JAVA <B>MonApplet</
B>.
</P>

    <APPLET   code="MonApplet.class"
              name="monApplet"
              width=600
```

```
                height=400>
    </APPLET>

</BODY>
</HTML>
```

La balise **<APPLET>** possède des propriétés. Voici la liste des propriétés de balise permises :

align permet de définir la position de l'***applet*** dans la page HTML. Elle peut être calée à gauche (valeur **left**), centrée (valeur **center**) ou calée à droite (valeur **right**).

alt permet de définir le texte qui serait affiché pour des navigateurs qui ne comprendraient pas JAVA.

code permet de définir quel fichier java est exécuté en tant qu'***applet***. Ce paramètre est obligatoire et le nom du fichier doit être un fichier **.class**. Le nom de ce fichier doit être le même que le nom de la classe de l'***applet***. (**MonApplet** pour l'exemple ci-dessus).

codebase permet de définir la localisation de l'***applet***, c'est-à-dire le répertoire dans lequel se trouve le fichier **.class** sur le serveur. Par défaut, le fichier **.class** est recherché dans le même répertoire que la page HTML.

height permet de définir la hauteur de l'***applet*** dans la page HTML. Cette dimension doit être exprimée en pixels.

name permet de d'identifier l'***applet*** en tant que composant de la page HTML. Ce paramètre peut être utilisé éventuellement dans de scripts.

width permet de définir la largeur de l'***applet*** dans la page HTML. Cette dimension doit être exprimée en pixels.

Pour une révision du langage HTML et l'utilisation des principales balises, on peut lire :

- **JAVA (Megapoche)**
  Le langage HTML : Chapitre 1

1.2.4 Affichage d'une applet

Une applet est une application graphique. Une telle application ne fonctionne pas du tout comme les applications en mode texte qui ont été réalisées jusqu'à présent.

Dans une application en mode texte, l'affichage se fait en séquence, à l'aide de la fonction **println** par exemple.

Une application graphique ne fonctionne jamais seule. Elle est visible dans une fenêtre et doit dessiner celle-ci à l'initialisation, bien sur, mais également chaque fois qu'il est nécessaire (lorsque son contenu a été caché momentanément par une autre fenêtre par exemple). A cette fin, elle reçoit un message **`paint`**.

Par défaut, la méthode **`paint`** de la classe **`Applet`** affiche un rectangle gris dont la taille est définie par les propriétés **`width`** et **`height`** de la balise **`<APPLET>`** dans la page HTML.

Ce fonctionnement par défaut, de façon évidente, ne convient pas. C'est pourquoi il faut surcharger cette méthode dans la classe de l'***applet*** que l'on est en train de développer.

1.2.5 Surcharge de la méthode paint

La méthode **`paint`** est la méthode qui est appelée chaque fois que l'***applet*** a besoin d'être redessinée. C'est une méthode virtuelle dans le sens où elle définit un comportement polymorphique pour la classe **`Applet`** et ses classes dérivées. Elle doit donc impérativement être déclarée avec la signature suivante :

```
public void paint(Graphics g)
{
    //...
}
```

La méthode **`paint`** reçoit en paramètre une instance de la classe **`Graphics`** qui définit le contexte d'affichage de l'applet. Pour dessiner l'applet il faut utiliser les méthodes de dessin de la classe **`Graphics`** comme, par exemple, la méthode **`drawString`** qui permet d'afficher du texte à un endroit précis de la fenêtre :

```
public void paint(Graphics g)
{
    g.drawString("Bonjour tout le monde !", 10, 20);
}
```

Les méthodes de dessin de la classe **`Graphics`** sont décrites dans l'ouvrage :

- **JAVA (Megapoche)**
  Affichage du texte : pages 252 à 267
  Méhodes de tracé : pages 268 à 301

1.3 TRAVAIL À RÉALISER

- Créer une nouvelle classe **`appPaint`** en dérivant la classe **`Applet`**. Ce nom a été choisi car cette ***applet*** permettra de dessiner, de façon similaire à un logiciel de dessin. Ces fonctionnalités seront ajoutées à l'***applet*** au fur et à mesure de l'avancée de ce cours.
- Surcharger la méthode **`paint`** de la classe de base pour que l'applet affiche le message "Bonjour tout le monde !".
- Créer la page HTML dans laquelle sera affichée l'***applet***. Tester cette page dans un navigateur capable de comprendre JAVA (Microsoft Explorer 4 par exemple).

FONCTIONS À UTILISER

Fonction	Langage JAVA	Classe	Package
Affichage de l'***applet***	`paint`	`Container`	`java.awt`
Affichage d'une chaîne de caractères	`drawString`	`Graphics`	`java.awt`
Classe **`Applet`**		`Applet`	`java.applet`

T.P. N°2 - PARAMÈTRES D'UNE APPLET

2.1 OBJECTIFS

- Paramétrage d'une applet en HTML
- Exploitation des paramètres en JAVA
- Cycle de vie d'une applet

2.2 CE QU'IL FAUT SAVOIR

2.2.1 Paramétrage d'une applet en HTML

Une applet JAVA peut être paramétrée dans la page HTML et ces paramètres externes peuvent être exploités dans le programme en JAVA de l'applet.

Les paramètres d'une applet sont définis par la balise **<PARAM>>**. Les balises **<PARAM>** doivent être placées entre les balises **<APPLET>** et **</APPLET>**. En HTML, il est possible de définir autant de balises **<PARAM>** que l'on veut. Les paramètres que l'applet ne sait pas gérée son ignorées sans générer d'erreur.

La balise **<PARAM>** possède deux propriétés :

name permet de définir le nom du paramètre. C'est un mot-clé, défini par le programmeur de l'applet, qui doit être connu par l'applet. Si un nom inconnu est utilisé, le paramètre est ignoré par l'applet sans générer d'erreur.

value permet de définir la valeur du paramètre. Cette valeur est une chaîne de caractères. Cependant des valeurs numériques peuvent être passée en paramètres par l'intermédiaire de chaînes de caractères ne contenant que des caractères numériques. Certains format conventionnel peuvent être aussi utilisés pour passer des paramètres particuliers comme une couleur en construisant une chaîne de caractères au format **"#RRVVBB"** ou **RR**, **VV** et **BB** sont respectivement les composantes rouge, verte et bleue de la couleur exprimées sur deux chiffres hexadécimaux.

Exemple :

```
<APPLET
   code="MonApplet.class"
   name="monApplet"
   width=300
   height=100>
   <PARAM NAME="background" VALUE="#44CC88">
   <PARAM NAME="sizeFont" VALUE="25">
   <PARAM NAME="message" VALUE="Bonjour tout le monde !">
</APPLET>
```

2.2.2 Exploitation des paramètres dans l'applet

A tout moment, dans la programmation de l'applet, il est possible de connaître la valeur d'un paramètre passé par la balise **<PARAM>** grâce à la méthode **getParameter** de la classe **Applet**. Le résultat de **getParameter** est de type **String**. Le paramètre de **getParameter** est une chaîne de caractères contenant le nom du paramètre tel qu'il a été indiqué dans la propriété de balise **name**. Le résultat de **getParameter** est une chaîne de caractères contenant la valeur indiquée dans la propriété de balise **value**.

```
String strParam = getParameter("background");
```

En supposant que la valeur indiquée pour le paramètre **background** dans la page HTML soit celle de l'exemple du paragraphe 2.2.1, la valeur de **strParam** obtenue serait **"#44CC88"**.

2.2.3 Propriétés de l'applet associées aux paramètres

Bien qu'il soit possible de connaître la valeur des paramètres à tout moment, l'usage veut que l'on associe une donnée membre de la classe de l'applet que l'on est en train de développer, à chaque paramètres. Il est pratique aussi de déclarer en constante, pour chaque paramètre, le nom du paramètre tel qu'il est indiqué dans la propriété **name** de la balise **<PARAM>** et la valeur par défaut que prend la donnée membre si l'utilisateur de l'applet omet la balise **<PARAM>** associée.

```
private final String PARAM_background = "background";
private final Color DEFAULT_background = new Color(255,255,255);
private Color m_background = DEFAULT_background;

private final String PARAM_sizeFont = "sizeFont";
private final int DEFAULT_sizeFont = 16;
private  int m_sizeFont = DEFAULT_sizeFont;

private final String PARAM_message = "message";
private final String DEFAULT_message = "Bonjour";
private  String m_message = DEFAULT_message;
```

Voici les conventions d'écriture :

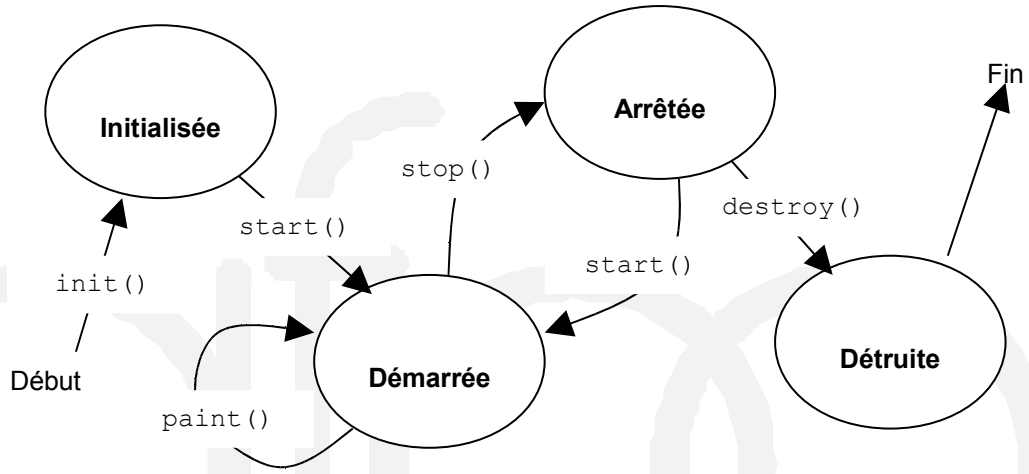
- Les propriétés associées aux paramètres sont préfixées **m_** et porte le même nom que le paramètre. Par exemple, la donnée membre associée au paramètre **background** est **m_background**. Elle est initialisée, lors de la déclaration à la valeur par défaut.

- La valeur par défaut est déclarée en constante (mot-clé **final**) et est préfixée **DEFAULT_**. Elle est de même type que la donnée membre et est utilisée pour initialiser cette dernière lors de sa déclaration.

- Le nom du paramètre est déclaré en constante et est préfixé **PARAM_**. La constante est de type **String** et initialisée à la même valeur que la chaîne de caractères figurant dans la propriété **name** de la balise **<PARAM>**.

2.2.4 Cycle de vie d'une applet

De sa naissance à sa mort une applet passe par plusieurs états. Chaque transition effectue un appel aux méthodes **`init`**, **`start`**, **`paint`**, **`stop`** et **`destroy`**. Ce cycle de vie peut être symbolisé par le diagramme ci-dessous.



La méthode init

La méthode **`init`** est la méthode qui est appelée une seule fois après que l'applet ait été chargée. Elle est utilisée pour initialiser les propriétés de l'applet. Elle est l'endroit privilégié pour initialiser les propriétés, associées aux paramètres transmis par la balise HTML **`<PARAM>`**, à l'aide de la méthode **`getParameter`**. C'est une méthode virtuelle dans le sens où elle définit un comportement polymorphique pour la classe **`Applet`** et ses classes dérivées. Elle doit donc impérativement être déclarée avec la signature suivante :

```
public void init()
{
    //...
}
```

La méthode start

La méthode **`start`** est la méthode qui est appelée une fois après que l'applet ait été chargée et initialisée. Elle peut être appelée plusieurs fois alternativement avec la méthode **`stop`** lorsque l'applet a été arrêtée et doit être redémarrée. Elle est l'endroit privilégié pour utilisée pour créer les ***threads*** dans un contexte multitâches.. C'est une méthode virtuelle dans le sens où elle définit un comportement polymorphique pour la classe **`Applet`** et ses classes dérivées. Elle doit donc impérativement être déclarée avec la signature suivante :

```
public void start()
{
    //...
    }
```

La méthode stop

La méthode **`stop`** est appelée plusieurs fois alternativement avec la méthode **`start`** lorsque l'applet doit être arrêtée (lorsque le navigateur est réduit en icône par exemple). Elle est l'endroit privilégié pour utilisée pour interrompre les ***threads*** qui ont été initialisées dans un contexte multitâches.. C'est une méthode virtuelle dans le sens où elle définit un comportement polymorphique pour la classe **`Applet`** et ses classes dérivées. Elle doit donc impérativement être déclarée avec la signature suivante :

```
public void stop()
{
    //...
}
```

La méthode destroy

La méthode **`destroy`** est la méthode qui est appelée une fois après que l'applet meurt (à la fermeture du navigateur par exemple). Comme la machine virtuelle Java est dotée d'un ramasse-miettes, son usage reste exceptionnel et semble faire double emploi avec la méthode **`stop`** . Mais comme la méthode **`finalize`** pour la classe **`Object`**, elle permet de contrôler plus finement la fermeture de l'applet. C'est une méthode virtuelle dans le sens où elle définit un comportement polymorphique pour la classe **`Applet`** et ses classes dérivées. Elle doit donc impérativement être déclarée avec la signature suivante :

```
public void destroy()
{
    //...
}
```

2.2.5 Initialisation des propriétés de l'applet

Toutes les propriétés associées aux paramètres transmis par **`<PARAM>`** n'ont pas la même nature et sont donc déclarées de types différents. La méthode **`getParameter`**. renvoie un résultat de type **`String`** qui doit être converti dans le type de la propriété. L'utilisateur de l'applet peut omettre certain paramètres ou bien passer des paramètres non compatibles avec le type de la propriété concernée et la conversion de ces paramètres peut générer des exceptions amenant un blocage de l'applet. C'est pourquoi quelques précaution sont indispensables.

Dans tous les cas les paramètres seront lus avec la fonction **`getParameter`** dans une variable de type **`String`**. Si l'utilisateur a omis le paramètre lu, **`getParameter`** renvoie **`null`** et on conserve la valeur par défaut utilisée lors de la déclaration de la propriété :

```
String strParam = getParameter(PARAM_message);
if (strParam != null)
{
    m_message = strParam;
}
```

Ceci est le cas le plus simple pour lequel aucune conversion n'est nécessaire. Pour les autres cas, des précautions sont indispensables et la gestion des exceptions s'impose. Voici les cas les plus couramment rencontrés :

Cas d'un paramètre numérique

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

La conversion d'une chaîne de caractères en entier de type **`int`** se fait en deux temps : d'abord la création d'une instance de la classe **`Integer`** construite à partir de la chaîne de caractères, puis la conversion en **`int`** par la méthode **`intValue`**. Si la chaîne de caractères contient des caractères non numériques, le constructeur de la classe **`Integer`** génère une exception de classe **`NumberFormatException`**. Ce qui provoque un branchement inconditionnel dans le bloc **`catch`** dans lequel cette exception est traitée (en fait, il n'y a rien à faire de particulier). La modification de la propriété est sautée. Elle conserve donc sa valeur par défaut initialisée à la déclaration.

Cas d'un paramètre de type couleur

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

La conversion d'une chaîne de caractères en objet de classe **`Color`** se fait par l'appel de la méthode **`stringToColor`**. Si la chaîne de caractère n'est pas au format standard HTML **`"#RRVVBB"`** ou **`RR`**, **`VV`** et **`BB`** sont respectivement les composantes rouge, verte et bleue de la couleur exprimées sur deux chiffres hexadécimaux, une exception de classe **`NumberFormatException`** est générée. Ce qui provoque un branchement inconditionnel dans le bloc **`catch`** dans lequel cette exception est traitée (en fait, il n'y a rien à faire de particulier). La propriété n'est pas modifiée. Elle conserve donc sa valeur par défaut initialisée à la déclaration.

Cela suppose qu'il existe une méthode qui permette la conversion d'une chaîne de caractères **`"#RRVVBB"`** en objet de classe **`Color`** qui puisse générer une exception si le format n'est pas respecté. Cette méthode n'existe pas en standard. En voici une implémentation possible :

```
private Color stringToColor(String paramValue)
   throws NumberFormatException
{
   try
```

```
        {
            int red =
            (Integer.decode("0x" + aramValue.substring(1,3))).intValue();
            int green =
            (Integer.decode("0x" + aramValue.substring(3,5))).intValue();
            int blue =
            (Integer.decode("0x" + aramValue.substring(5,7))).intValue();

            return new Color(red,green,blue);
        }
        catch (StringIndexOutOfBoundsException e)
        {
            throw new NumberFormatException();
        }
    }
```

Les caractères RR, VV, BB sont successivement isolés par la méthode **substring** et convertis de l'hexadécimal en entier par la méthode **decode**. Si il manque des caractères à la chaîne **"#RRVVBB"**, **substring** génère une exception de classe **StringIndexOutOfBoundsException**. Si les caractères de la chaîne de sont pas des caractères hexadécimaux, génère une exception de classe **NumberFormatException**. Il faut donc convertir l'exception de classe **StringIndexOutOfBoundsException** en exception de classe **NumberFormatException** pour n'avoir qu'une seule classe d'exception à gérer dans le programme appelant. Pour cela il suffit de surveiller (**try**) ces conversion et de capturer (**catch**) l'exception de classe **StringIndexOutOfBoundsException** pour y générer une autre exception de classe **NumberFormatException**.

Cas de paramètres à plusieurs composantes séparées par des virgules

Ce type de paramètres peut être composé de plusieurs mots-clés séparés par des virgules, dont on ne connaît a priori ni le nombre, ni l'ordre.

C'est le cas pour le style de police, par exemple, ou on utilise les mots-clés **bold** (caractères gras) ou **italic** (caractères italiques) dans n'importe quel ordre en les séparant par un virgule s'ils sont utilisé tous les deux.

```
String strParam = getParameter(PARAM_styleFont);
if (strParam != null)
{
    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)
```

```
        {
        }
    }
```

Comme on ne connaît pas le nombre de mots utilisés, on fait une boucle sans fin (condition **true**). Cette boucle est surveillée dans un bloc **try** et interrompue dès qu'une exception est générée. A chaque itération, on utilise la méthode **fieldOf** qui permet d'extraire un champ d'une chaîne de caractères. **fieldOf** génère une exception de classe **IndexOutOfBoundsException** si le numéro de champ est inférieur à 1 ou supérieur au nombre de champs contenus dans la chaîne. Chaque champ est comparé à l'un des mots-clés (**bold** ou **italic**). Et, en cas d'égalité, le bit correspondant de la propriété **m_styleFont** est activé.

Malheureusement, la méthode **fieldOf** n'existe pas en standard. En voici une implémentation possible[1] :

```
private String fieldOf(String paramValue, int nField, int Sep)
    throws IndexOutOfBoundsException
{
    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)
       {
              strResult = paramValue.substring(nDebut);
              nDebut = paramValue.length() + 1;
       }
       else
       {
          strResult = paramValue.substring(nDebut, nFin);
          nDebut = nFin + 1;
       }
    }
    return strResult;
}
```

Si le N° de champ **nField** est inférieur à 1, une exception de classe **IndexOutOfBoundsException** est générée par **throw**. Si le N° de champ est supérieur au nombre de champs contenu dans **paramValue**, **substring** va générer une exception de classe **StringIndexOutOfBoundsException**. Cette classe est elle-même dérivée de **IndexOutOfBoundsException**, ce qui permet de ne capturer que ce dernier type d'exception dans les programme appelants.

2.2.6 Surcharge de la méthode getParameterInfo

La méthode **getParameterInfo** est une méthode utilisée par les environnements de développement pour obtenir des informations sur l'applet utilisée et les paramètres qu'elle accepte. Elle doit fournir en résultat un tableau de chaînes de caractères à 3 colonnes dont les contenus sont respectivement le nom des paramètres (ceux passé par la propriétés **name** de la balise **<PARAM>**), leur type (String, Integer, Color, etc.) et leur description sommaire. C'est une méthode virtuelle dans le sens où elle définit

[1] Ctte méthode a fait l'objet du T.P. N° 5 du support JAVA_AFS.

un comportement polymorphique pour la classe **`Applet`** et ses classes dérivées. Elle doit donc impérativement être déclarée avec la signature suivante :

```
public String[][]getParameterInfo()
{
    String[][] info =
    {
        { PARAM_background, "String",
              "Couleur d'arrière-plan, format \"#rrggbb\"" },
        { PARAM_sizeFont, "Integer",
              "Corps de la police de caractères" },
        { PARAM_message, "String",
              "Message à afficher dans l'applet" }
        };
    return info;
}
```

Bien que cette méthode ne soit pas utilisée par l'applet elle-même, le fait qu'elle soit surcharger pour fournir des informations précises sur l'applet permet de l'intégrer dans les outils utiliser pour créer des pages WEB en générant automatiquement le code HTML.

2.3 TRAVAIL À RÉALISER

A partir du T.P. précédent modifier l'applet pour qu'elle affiche un message quelconque avec une police paramétrable. Les paramètres seront les suivants :

`background` : Couleur d'arrière-plan, format **"#RRVVBB"**,
`foreground` : Couleur d'affichage, format **"#RRVVBB"**,
`font` : Nom de la police de caractère **"Helvetica"**, **"TimesRoman"**, **"Courrier"**, **"Dialog"** ou **"ZapfDingbats"**
`sizeFont` : Corps de la police de caractères (entier),
`styleFont` : Style de la police de caractères bold (gras) ou/et italique (italic),
`message` : Message à afficher dans l'applet.

Pour cela :

- Surcharger la méthode **`init`** pour qu'elle initialise correctement des propriétés de l'applet associées aux paramètres passé par la balise **`<PARAM>`** dans la page HTML.
- Surcharger la méthode **`paint`** de la classe de base pour que l'applet affiche le message avec les attributs passés en paramètre pour qu'il soit centré.
- Surcharger la méthode **`getParameterInfo`** de la classe de base pour que l'applet fournisse les informations correctes sur les paramètres qu'elle accepte.
- Créer la page HTML dans laquelle sera affichée l'***applet***. Tester cette page dans un navigateur capable de comprendre JAVA (Microsoft Explorer 4 par exemple).

FONCTIONS À UTILISER

Fonction	Langage JAVA	Classe	Package
Affichage de l'***applet***	`paint`	`Container`	`java.awt`
Affichage d'une chaîne de caractères	`drawString`	`Graphics`	`java.awt`
Changement de police de caractères	`setFont`	`Graphics`	`java.awt`
Changement de couleur d'écriture	`setColor`	`Graphics`	`java.awt`
Calcul des informations numériques sur une police	`getFontMetrics`	`Component`	`java.awt`
Calcul de l'espace occupé par une chaîne	`stringWidth`	`FontMetrics`	`java.awt`
Obtention de la valeur d'un paramètre <PARAM>	`getParameter`	`Applet`	`java.applet`
Obtention de la taille de l'applet sur la page HTML	`getSize`	`Component`	`java.awt`
Classe **`Applet`**		`Applet`	`java.applet`
Classe **`Font`**		`Font`	`java.awt`
Classe **`FontMetrics`**		`FontMetrics`	`java.awt`
Classe **`Color`**		`Color`	`java.awt`
Classe **`Integer`**		`Integer`	`java.lang`

T.P. N°3 - INTERACTIONS UTILISATEURS

3.1 OBJECTIFS

- Messages du clavier
- Messages de la souris

3.2 CE QU'IL FAUT SAVOIR

3.2.1 Les interactions utilisateurs

L'utilisation d'une applet dans une page HTML n'a de sens que s'il faut créer des interactions utilisateurs sur une page HTML, fonctionnalité qui n'existe pas en standard dans le langage HTML.

Les interactions utilisateurs peuvent être classées en trois groupes :

- les manipulations de la souris,
- la frappe de touches au clavier,
- l'acquisition de données à travers des objets d'interface comme les boutons, les cases à cocher, les zones de saisie, etc.

Ce chapitre ne traite que des deux premiers groupes, le troisième sera abordé dans le chapitre suivant.

3.2.2 Les messages du clavier

A chaque fois que l'utilisateur tape une touche aux clavier, lorsque l'applet est active, l'applet reçoit des messages qu'elle peut traiter par l'intermédiaire de méthodes.

La méthode keyDown

La méthode **`keyDown`** est appelée à chaque fois que l'utilisateur appuie sur une touche du clavier. Lorsque l'utilisateur maintient un touche longtemps appuyée, plusieurs messages **keyDown** sont envoyés. Ce qui permet traiter la répétition. C'est une méthode virtuelle dans le sens où elle définit un comportement polymorphique pour la classe **`Applet`** et ses classes dérivées. Elle doit donc impérativement être déclarée avec la signature suivante :

```
public boolean keyDown(Event evt, int key)
{
    //...
}
```

- **`evt`** est l'événement qui est généré. Ce paramètre donne informations sur les touches joker qui sont utilisée comme MAJ, CTRL ou ALT.

```
public boolean keyDown(Event evt, int key)
{
    if (evt.shiftDown())
    {
        // La touche MAJ est enfoncée
    }
    if (evt.controlDown())
    {
```

```
            // La touche CTRL est enfoncée
        }
        if (evt.metaDown())
        {
            // La touche ALT est enfoncée
        }
        //...
    }
```

- **key** est le code ASCII de la touche pressée. S'il s'agit d'une touche spéciale, **key** prend la valeur d'un code spécial.

Variable	Touche	Valeur
Event.ESC	Echap	27
Event.INSERT	Insère	1025
Event.DELETE	Supprime	127
Event.HOME	Origine	1000
Event.UP	Flèche vers le haut	1004
Event.PGUP	Page précédente	1002
Event.LEFT	Flèche vers la gauche	1006
Event.RIGHT	Flèche vers la droite	1007
Event.END	Fin	1001
Event.DOWN	Flèche vers le bas	1005
Event.PGDN	Page suivante	1003
Event.F1	Fonctions F1	1008
à	à	à
Event.F12	F12	1019

La méthode keyUp

La méthode **keyUp** est appelée lorsque l'utilisateur relâche sur une touche du clavier. Lorsque l'utilisateur maintient un touche longtemps appuyée, plusieurs messages **keyDown** peuvent être envoyés, mais **keyUp** n'est envoyé qu'une fois. Ce qui permet traiter la répétition. C'est une méthode virtuelle dans le sens où elle définit un comportement polymorphique pour la classe **Applet** et ses classes dérivées. Elle doit donc impérativement être déclarée avec la signature suivante :

```
public boolean keyUp(Event evt, int key)
{
    //...
}
```

Les deux méthodes **keyDown** et **keyUp** doivent renvoyer un résultat booléen, vrai si le message est assumé par l'applet, faux si le message doit être traité par défaut tel qu'il est implémenté dans la classe parente.

Sauf pour un usage particulier, on utilise peu ces méthodes pour saisir du texte. On préfère utiliser des objets d'interface comme les zones de saisies. Leur utilisation sera traitée dans le chapitre suivant.

3.2.3 Les messages de la souris

A chaque fois que l'utilisateur effectue une manipulation de souris lorsque l'applet est active, que ce soit le click des boutons ou un déplacement, l'applet reçoit des messages qu'elle peut traiter par l'intermédiaire de méthodes.

La méthode mouseDown

La méthode **mouseDown** est appelée lorsque l'utilisateur presse le bouton de la souris. C'est une méthode virtuelle dans le sens où elle définit un comportement polymorphique pour la classe **Applet** et ses classes dérivées. Elle doit donc impérativement être déclarée avec la signature suivante :

```
public boolean mouseDown(Event evt, int x, int y)
{
    //...
}
```

- **evt** est l'événement qui est généré. Ce paramètre donne informations sur les touches joker qui sont utilisée comme MAJ, CTRL ou ALT.

```
public boolean mouseDown(Event evt, int x, int y)
{
    if (evt.shiftDown())
    {
        // La touche MAJ est enfoncée
    }
    if (evt.controlDown())
    {
        // La touche CTRL est enfoncée
    }
    if (evt.metaDown())
    {
        // La touche ALT est enfoncée
    }
    //...
}
```

- **x** et **y** définissent la position du curseur de la souris à partir du coin supérieur gauche de l'applet..

La méthode mouseUp

La méthode **mouseUp** est appelée lorsque l'utilisateur relâche le bouton de la souris. C'est une méthode virtuelle dans le sens où elle définit un comportement polymorphique pour la classe **Applet** et ses classes dérivées. Elle doit donc impérativement être déclarée avec la signature suivante :

```
public boolean mouseUp(Event evt, int x, int y)
{
    //...
}
```

La méthode mouseMove

La méthode **mouseMove** est appelée lorsque l'utilisateur déplace la souris et que le bouton de la souris n'est pas enfoncé. C'est une méthode virtuelle dans le sens où elle définit un comportement polymorphique pour la classe **Applet** et ses classes dérivées. Elle doit donc impérativement être déclarée avec la signature suivante :

```
public boolean mouseMove(Event evt, int x, int y)
{
    //...
}
```

La méthode mouseDrag

La méthode **mouseDrag** est appelée lorsque l'utilisateur déplace la souris et que le bouton de la souris est enfoncé. C'est une méthode virtuelle dans le sens où elle définit un comportement polymorphique pour la classe **Applet** et ses classes dérivées. Elle doit donc impérativement être déclarée avec la signature suivante :

```
public boolean mouseDrag(Event evt, int x, int y)
{
    //...
}
```

La méthode mouseEnter

La méthode **mouseEnter** est appelée lorsque l'utilisateur déplace la souris et que le curseur entre dans la zone graphique occupée par l'applet. C'est une méthode virtuelle dans le sens où elle définit un comportement polymorphique pour la classe **Applet** et ses classes dérivées. Elle doit donc impérativement être déclarée avec la signature suivante :

```
public boolean mouseEnter(Event evt, int x, int y)
{
    //...
}
```

La méthode mouseExit

La méthode **mouseExit** est appelée lorsque l'utilisateur déplace la souris et que le curseur sort de la zone graphique occupée par l'applet. C'est une méthode virtuelle dans le sens où elle définit un comportement polymorphique pour la classe **Applet** et ses classes dérivées. Elle doit donc impérativement être déclarée avec la signature suivante :

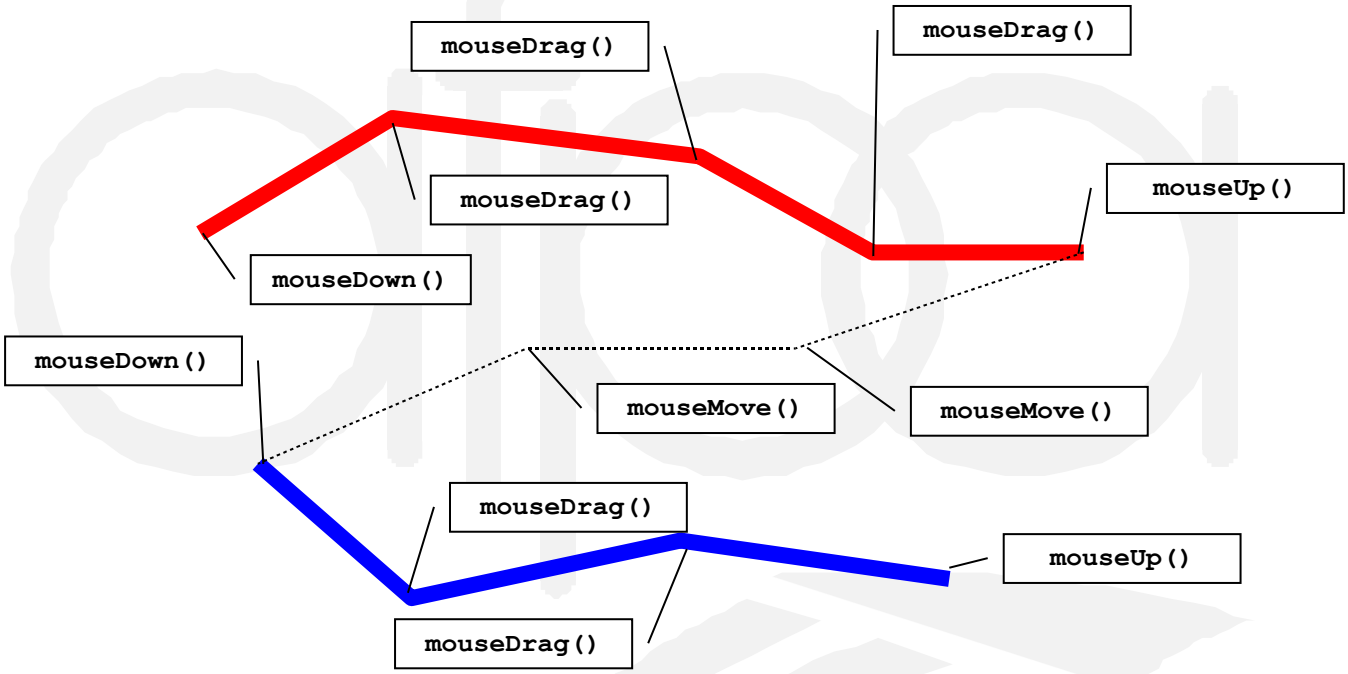
```
public boolean mouseExit(Event evt, int x, int y)
{
    //...
}
```

Toutes ces méthodes doivent renvoyer un résultat booléen, vrai si le message est assumé par l'applet, faux si le message doit être traité par défaut tel qu'il est implémenté dans la classe parente.

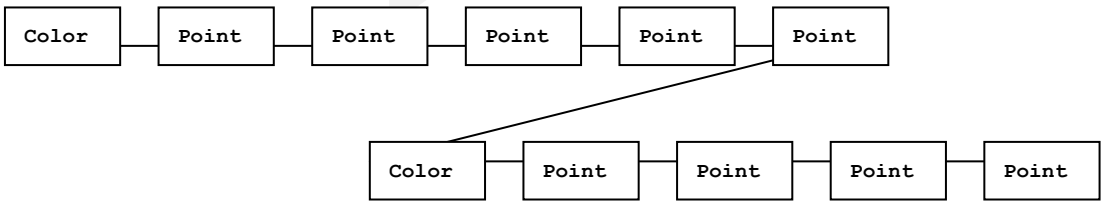
3.2.4 Traitement des messages de la souris

Dans les deux premiers T.P., on à perdu de vue l'objectif final qui est de créer une applet permettant de dessiner, de façon similaire aux applications graphiques de Windows, sur une page HTML.

Pour ce type d'application, ce sont bien sur les messages de la souris qui sont le plus intéressants à traiter. Le dessin ci-dessous illustre le lien qu'il va falloir faire entre les actions de l'utilisateur et le traitement des messages :



Tous ces messages renseignent l'applet sur la position du curseur de la souris. Il suffit donc à l'applet de ranger toutes ces positions dans une collection de classe **`Vector`**. Pour identifier chaque trait de crayon et ne pas dessiner le lien virtuel entre la fin de l'un et le début de l'autre, il suffit d'intercaler dans la collection un objet d'une autre classe, la classe **`Color`** par exemple. Ce qui permet de ranger et d'identifier des traits de couleurs différentes :



3.3 TRAVAIL À RÉALISER

A partir du T.P. précédent modifier l'applet pour qu'elle permet à un utilisateur de dessiner dessus. On ne conservera que les paramètres suivants :

`background` : Couleur d'arrière-plan, format **"#RRVVBB"**,
`foreground` : Couleur d'affichage, format **"#RRVVBB"**.

Pour cela :

- Surcharger les méthodes adéquates de la souris pour qu'elles permettent d'enregistrer des points dans une collection de classe **`Vector`**.
- Ne pas oublier de modifier la méthode **`paint`** pour que l'applet affiche le dessin enregistré dans la collection en faisant une itération de cette dernière.
- Surcharger les méthodes adéquates du clavier pour que l'applet s'efface en appuyant la touche **Suppr**.
- Créer la page HTML dans laquelle sera affichée l'***applet***. Tester cette page dans un navigateur capable de comprendre JAVA (Microsoft Explorer 4 par exemple).

FONCTIONS À UTILISER

Fonction	Langage JAVA	Classe	Package
Messages du clavier	`keyUp` `keyDown`	`Component`	`java.awt`

Messages de la souris	`mouseDown`
	`mouseUp`
	`mouseEnter`
	`mouseExit`
	`mouseMove`
	`mouseDrag`
		`Component`	`java.awt`

Affichage de l'***applet***	`paint`	`Container`	`java.awt`
Déclenchement de l'affichage de l'***applet***	`repaint`	`Component`	`java.awt`
Tracé d'un trait entre deux points	`drawLine`	`Graphics`	`java.awt`
Changement de couleur d'écriture	`setColor`	`Graphics`	`java.awt`
Classe **`Applet`**		`Applet`	`java.applet`
Classe **`Color`**		`Color`	`java.awt`
Classe **`Vector`**		`Vector`	`java.util`

CONCLUSION

Ici se termine la première partie de ce cours. Il est difficile d'aller plus loin dans la programmation en JAVA sans aborder les concepts de la Programmation Orientée Objet. Ces concepts feront l'objet de la deuxième partie de ce cours.

INDEX ANALYTIQUE