

Développement d'un composant Delphi

Mise en œuvre : Delphi 6 professionnel

A/ Eléments de base

1. Présentation de l'application

Il s'agit de programmer une horloge analogique avec ses trois aiguilles. Ce type de programme peut être facilement réutilisé par toute application nécessitant un affichage de l'heure courante. Nous allons donc programmer cette horloge sous la forme d'un composant qui sera intégré à la palette Delphi.

2. Un premier pas

Le premier pas à accomplir est la mise au point (c'est le cas de le dire) de la classe représentée ci-dessous :

Tpoint
abscisse ordonnée
getAbs() getOrd() setPos(abscisse, ordonnée) déplacer(deltaX, deltaY)

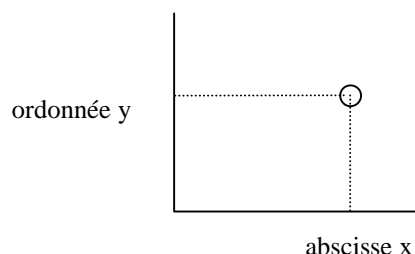
Cette classe est destinée à représenter un point sur l'écran. Elle nous sera utile pour matérialiser les extrémités des aiguilles par exemple.

Travail à faire

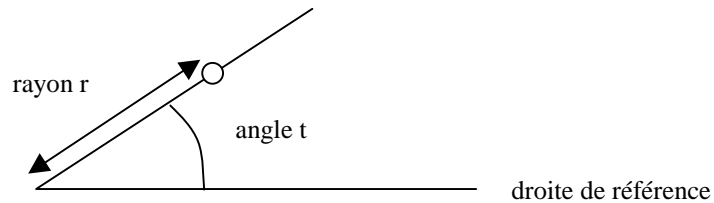
- Créez un nouveau projet Delphi, munissez-le d'une fiche et d'une unité "Point" contenant la description de cette classe.
- Testez le bon fonctionnement en mettant des boutons de commande sur la fiche permettant d'initialiser, de déplacer, et d'afficher les coordonnées d'un point.

3. Points polaires

Les objets de la classe Tpoint sont représentés en coordonnées cartésiennes, c'est-à-dire exprimées en référence à un repère constitué par deux axes comme illustré ci-dessous.



Il existe un autre système de repérage d'un point dans un plan : les coordonnées polaires. Le point est cette fois caractérisé par un angle (par rapport à une droite de référence) et un rayon (distance en le centre du repère et le point).



Le passage d'un système de coordonnées à l'autre est assez simple :

> cartésiennes (x,y) \rightarrow polaires (r,t) :

$$r \leftarrow \text{racine carrée} (x^2 + y^2)$$

t est défini par l'algorithme suivant :

```

si x=0 alors
  si y=0 alors
    t  $\leftarrow$  0
  sinon
    si y>0 alors
      t  $\leftarrow$  pi / 2
    sinon
      t  $\leftarrow$  - pi / 2
    fin de si
  fin de si
sinon
  si y>0 alors
    t  $\leftarrow$  arc tangente ( y / x )
  sinon
    t  $\leftarrow$  arc tangente ( y / x ) + pi
  fin de si
fin de si

```

> polaires (r,t) \rightarrow cartésiennes (x,y) :

$$x \leftarrow r * \cosinus (t)$$

$$y \leftarrow r * \sinus (t)$$

Nous allons créer une classe TpointPolaire, dérivée de Tpoint, ne possédant aucun membre donnée supplémentaire mais permettant :

- D'initialiser un point avec des coordonnées polaires.
- De fournir la coordonnée polaire rayon (r).
- De fournir la coordonnée polaire angle (t).

Jeu d'essai :

Le point de coordonnées polaires (r=10, t=1) correspond au point cartésien (x=5, y=8).

Le point de coordonnées polaires (r=10, t=2) correspond au point cartésien (x=-4, y=9).

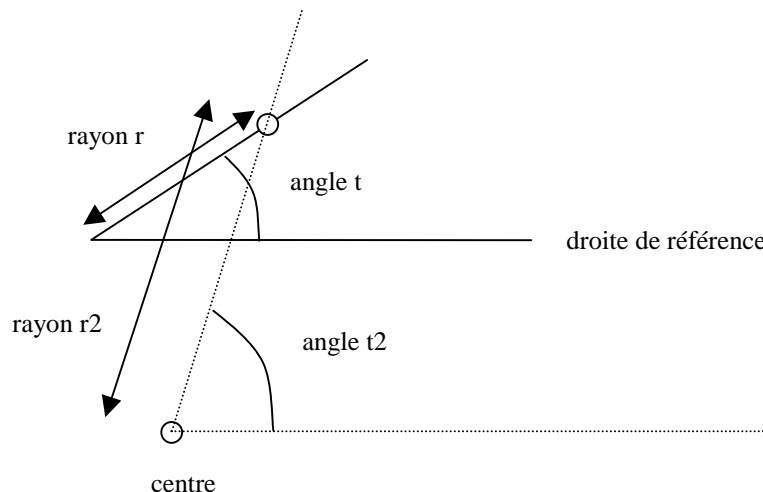
Remarque : l'angle t est exprimé en radians.

Travail à faire

- Complétez le diagramme de classes présenté plus haut.
- Ajoutez la classe TpointPolaire à l'unité Point.pas et testez-en le fonctionnement.
- Complétez votre classe en ajoutant la méthode suivante :

```
procedure TpointPolaire.setPosPolaireCentre(p_r, p_t : double ; p_centre : TPoint);  
begin  
    setPosPolaire(p_r,p_t);  
    deplacer(p_centre.getAbs(),p_centre.getOrd());  
end;
```

Cette méthode permettra de gérer des points polaires exprimés en fonction d'un centre de repère particulier.



A l'issue de l'exécution de setPosPolaireCentre, le point contiendra les coordonnées cartésiennes exprimées par rapport au centre passé en paramètre (centre du repère). La lecture de ses coordonnées polaires donnera donc comme résultat r2 et t2.

4. Création du composant

Chaque élément de la Visual Component Library (VCL) de Delphi est un composant utilisable de manière simple dans toute application (il suffit de le sélectionner dans la palette où il apparaît).

Chaque composant fait partie d'un paquet de composants, un paquet pouvant en regrouper plusieurs.

D'autre part, chaque composant doit être enregistré pour apparaître dans l'interface : c'est le rôle de la procédure "registerComponent" qui précise dans quelle page de palette le composant doit être installé.

La démarche générale de création d'un nouveau composant est la suivante :

- Créer un nouveau paquet pour contenir le composant (ou ouvrir un paquet existant pour l'y ajouter). La création d'un nouveau paquet se fait simplement par "fichier/nouveau/autre/paquet".
- Créer un nouveau composant (composants/nouveau composant).
- Indiquer la classe dont dérive le composant, le nom de la classe du composant, la page de palette de destination et le nom de l'unité contenant le code.
- Mettre au point le composant en intégrant un objet de ce type directement dans un projet.
- Installer le composant dans la palette.

Travail à faire

- Conservez votre projet ouvert, créez un nouveau paquet et enregistrez-le sous le nom "paquetbases.dpk".
- Ajoutez un composant "Thorloge" à ce paquet : il dérive de TgraphicControl (controls) et doit s'installer dans la page de palette "exemples". Son source se situe dans l'unité "horloge.pas".
- Examiner le code généré, on retrouve bien la procédure d'enregistrement du composant.
- Modifiez le code de la classe Thorloge en ajoutant une méthode publique "Paint"

Déclarée comme suit :

```
procedure Paint; override;
```

Et définie comme suit :

```
procedure THorloge.Paint;  
// appelée automatiquement par le système en cas de réaffichage du composant  
begin  
    canvas.Pen.Width:=1;  
    canvas.Ellipse(0,0,Width,Height);  
end;
```

- Compiler le paquet.
- Ajouter "horloge.pas" à la clause uses de la fiche du projet, déclarez-y un membre donnée privé "monHorloge" de type Thorloge et prévoyez son initialisation et sa destruction :

Initialisation (sur l'événement FormCreate) :

```
monHorloge:=THorloge.Create(Self);  
monHorloge.Parent:=Self;  
monHorloge.Width:=151;  
monHorloge.Height:=151;  
monHorloge.Top:=100;  
monHorloge.Left:=100;
```

Destruction (sur l'événement FormDestroy)

```
monHorloge.Free;
```

- Testez l'application

5. Mais comment ça marche ?

> Le code d'initialisation et de destruction du composant Thorloge n'est utile qu'en phase de test. En effet, lorsque le composant sera installé :

- La création de la fiche entraînera automatiquement celle de l'horloge.
- Le programmeur qui l'utilise fixera sa taille et sa position à l'aide de la souris en phase de conception.
- La destruction de la fiche entraînera celle de l'horloge.

> La méthode Paint (héritée de TgraphicControl) est appelée automatiquement lorsque le composant a besoin d'être redessiné (lorsque l'utilisateur déplace la fenêtre par exemple). Il suffit donc de redéfinir cette méthode virtuelle pour indiquer comment le contrôle doit être représenté.

> L'objet Canvas est en fait une surface de dessin. Cette propriété héritée de TgraphicControl fournit tout ce qui est nécessaire pour dessiner : un crayon (Pen), des routines graphiques (comme Ellipse), ...

Travail à faire

- Refaites la même chose en essayant de ne pas trop regarder cette page...

6. Installation du composant

> Avant d'installer le composant dans la palette, il est préférable de le munir d'une icône. Cette icône, qui apparaîtra en mode conception, peut être facilement réalisée grâce à l'outil intégré "Editeur d'images".

Le principe est le suivant :

- On crée un fichier de "ressources composants" portant le nom de l'unité : "horloge.dcr".
- Dans ce fichier, on crée autant de ressources de type bitmap qu'il y a de composants à recenser (ici un seul).
- Chaque bitmap doit mesurer 24 pixels sur 24 pixels et porter le nom du composant concerné (ici Thorloge).
- L'association entre le fichier dcr et le paquet de composant est fait en indiquant au début du fichier unité (ici horloge.pas) une directive de ressource sous la forme : {\$R Horloge.dcr}.

Remarque : si le fichier ".dcr" existe lors de l'ajout du fichier ".pas" au projet, il est automatiquement inclus dans le paquet et la directive \$R devient inutile.

> Une fois le composant testé, son icône prête, il n'y a plus qu'à procéder à l'installation (après compilation du paquet). En fait, on n'installe pas un composant mais un paquet, ce qui signifie que tous les composants d'un même paquet sont installés ou désinstallés en même temps. Il est possible de masquer certains composants pour ne pas encombrer la palette, mais ils ne sont pas désinstallés pour autant.

Les options du paquet permettent d'indiquer une description qui apparaîtra dans le gestionnaire de paquets ("Les bases d'une horloge analogique" par exemple).

Deux méthodes peuvent être utilisées pour l'installation :

- Ouvrir le paquet, le compiler puis cliquer sur installer.
- Choisir l'option "Installer des paquets" du menu "Composant", cliquer sur ajouter, sélectionner le fichier d'extension ".bpl" associé au paquet et cliquer sur le bouton "ouvrir".

Dans les deux cas, le (ou les) composant(s) du paquet se retrouve(nt) dans la palette, sur la page prévue par la routine de recensement.

La désinstallation du paquet se fait à l'aide de l'option "Installer des paquets" du menu "Composant", en sélectionnant le paquet et en cliquant sur le bouton "retirer".

> En cas de problème...

Si votre composant ne fonctionne pas comme vous l'espériez, le plus simple est de désinstaller le paquet et de reprendre les tests avant installation. Si ces tests s'avèrent corrects, c'est l'installation du paquet qui se passe mal... reprenez-la calmement !

Travail à faire

- Créez un fichier ressource pour votre paquet contenant le bitmap suivant pour l'horloge.



- Installez le paquet par la méthode de votre choix.
- Testez le composant à l'aide d'un nouveau projet.
- Désinstallez le paquet.

B/ Une horloge simple

1. Les classes

Dans cette seconde étape, nous allons réaliser un composant horloge "basique". Le résultat attendu est une horloge analogique munie de trois aiguilles et indiquant en permanence l'heure exacte. Voici la description des éléments permettant de la réaliser.

L'horloge :

Une horloge (classe dérivée de TgraphicControl) possède les caractéristiques suivantes :

- Un centre, il s'agit d'un point permettant de situer l'horloge sur la fenêtre.
- Un rayon, rayon du cadran de l'horloge exprimé en pixels.
- Trois aiguilles, indiquant respectivement l'heure, la minute et la seconde courante.
- Un timer (appelons-le "temps"), il s'agit d'un composant Delphi capable de générer un événement à intervalle régulier. Il sera utilisé pour faire "avancer" l'horloge.

Les aiguilles :

Une aiguille est définie par :

- Un point origine matérialisant l'emplacement où l'aiguille est fixée (le centre de l'horloge en fait).
- Un point extrémité, ce point étant recalculé à chaque déplacement de l'aiguille. Il sera judicieux d'utiliser un point polaire pour représenter cette extrémité.
- Des caractéristiques de forme : longueur, épaisseur, couleur.
- Une surface de dessin, en fait le canevas sur lequel l'aiguille doit se dessiner. Ce canevas lui sera fourni par l'horloge (celle-ci en hérite un de TgraphicControl).

Travail à faire

- Représentez graphiquement les classes nécessaires à la réalisation de l'horloge.

2. Réalisation du composant

Classe Thorloge (dans le fichier horloge.pas) :

```
THorloge=class(TGraphicControl)
// composant graphique "horloge analogique"
private
    centre:TPoint;           // point d'accroche des aiguilles
    rayon:integer;           // rayon de l'horloge en pixels
    heure,minute,seconde:TAiguille; // les trois aiguilles de l'horloge
    temps:TTimer;            // déclencheur d'avancement
    procedure init;           // appelée au début, au redimensionnement
    procedure dessinerCadran; // dessin du cadran de l'horloge
    procedure tempsTimer(sender : TObject); // procédure déclenchée chaque seconde par le timer
    procedure getMoment(var h,m,s:double); // renseigne h,m,s:position des aiguilles selon l'heure courante
public
    constructor Create(AOwner:TComponent);override; // constructeur redéfini
    procedure Paint;override; // appelée par le système pour rafraîchir l'affichage du composant
    destructor Destroy;override; // destructeur redéfini
end;
```

Remarques :

- Le constructeur doit entre autres créer tous les objets possédés par l'horloge, le destructeur les détruira.
- La méthode init affecte la procédure tempsTimer à l'événement OnTimer du timer temps, calcule le centre et le rayon de l'horloge, et initialise les trois aiguilles.
- La procédure tempsTimer appelle chaque seconde la procédure getMoment, puis repositionne chacune des aiguilles par un appel à la méthode setPosition de la classe aiguille.
- Les fonctions Now, HourOf, MinuteOf et SecondOf sont utiles pour la réalisation de getMoment.

Classe Taiguille (dans le fichier horloge.pas) :

```
TAiguille=class(TObject)
// une aiguille d'horloge
private
    canvas:TCanvas;    // surface de dessin de l'aiguille (fournie par l'horloge)
    origine:TPoint;    // point d'accroche de l'aiguille
    extremite:TPointPolaire; // ce point est modifié à chaque déplacement de l'aiguille
    longueur,epaisseur:integer; // permet la distinction visuelle heure, minute et seconde
    couleur:TColor;    // couleur de dessin de l'aiguille (toujours noir dans cette version)
    procedure setPosition(p_position:double);
        // p_position compris entre 0 et 60, double pour éviter les "sauts"
    procedure dessiner(p_couleur:TColor); // affichage ou effacement de l'aiguille selon la couleur
public
    constructor Create(p_canvas:TCanvas);overload; // surcharge (signature différente)
    procedure init(p_origine:TPoint;p_longueur,p_epaisseur:integer;p_couleur:TColor);
        // appelée à l'initialisation de l'horloge
    destructor Destroy;override; // redéfinition (signature identique)
end;
```

Travail à faire

- Schématisez le fonctionnement de l'horloge décrit ci-dessous à l'aide d'un diagramme de séquence.

Chaque seconde, le timer "temps" envoie un message à l'horloge et appelle sa méthode "tempsTimer". Cette méthode appelle "getMoment" pour récupérer l'heure système, puis fait appel à la méthode setPosition de la classe Taiguille trois fois de suite : pour l'aiguille "seconde", pour l'aiguille "minute", puis pour l'aiguille "heure" (on ne s'intéressera ici qu'à l'aiguille des secondes).

L'aiguille qui reçoit le message "setPosition" appelle la méthode dessiner avec la couleur de fond (cet appel efface donc l'aiguille), puis envoie le message setPosPolaireCentre à l'objet "extrémité" (ce qui repositionne l'extrémité de l'aiguille), et enfin appelle une seconde fois la méthode dessiner avec la couleur noire (ce qui réaffiche l'aiguille à sa nouvelle place).

- Complétez le fichier horloge.pas en y ajoutant la déclaration des classes Thorloge et Taiguille.
- Définissez chacune des méthodes prévues dans la partie implémentation de l'unité.
- Testez votre composant.

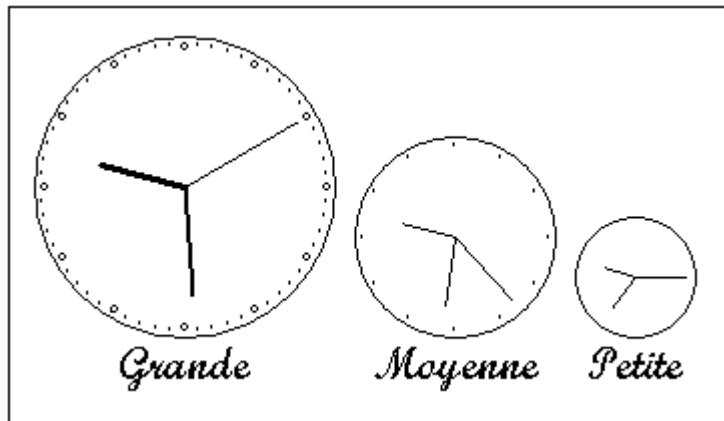
C/ Amélioration du composant

1. Trois tailles de cadran

L'utilisateur du composant horloge détermine graphiquement sa taille en phase de conception. En fonction de cette taille, nous allons déterminer un niveau de détail pour l'affichage de l'horloge.

- Taille 1 (petite, rayon ≤ 35 pixels) : les aiguilles sont toutes les trois d'épaisseur 1 et le cadran est matérialisé par un simple cercle.
- Taille 2 (moyenne, rayon compris entre 35 et 60 pixels) : les aiguilles sont toutes d'épaisseur 1 et le cadran indique les heures.
- Taille 3 (grande, rayon > 60 pixels) : les aiguilles sont d'épaisseurs différentes (1, 2 et 3 pixels) et le cadran indique les heures et les minutes.

Le dessin ci-dessous résume les trois cas :



Le principe de réalisation est le suivant :

- La taille de l'horloge est calculée à son initialisation, elle est mémorisée en partie privée et les aiguilles sont initialisées en fonction de cette information.
- Le dessin du cadran varie en fonction de la taille.

Travail à faire

- Modifiez votre composant pour y ajouter la notion de taille.

2. Affichage de la date

En taille 3, l'horloge est suffisamment grande pour permettre l'affichage de la date du jour sous la forme JJ/MM/AAAA dans la partie inférieure du cadran. :



- L'emplacement d'affichage de la date peut être calculé et mémorisé à l'aide d'un point lors de l'initialisation de l'horloge.
- Le code suivant place la date formatée dans une variable jour de type string :

```
moment:=Now;  
jj:=DayOf(moment);  
mm:=MonthOf(moment);  
aa:=YearOf(moment);  
jour:=format('%2d/%2d/%4d',[jj,mm,aa]);
```

Travail à faire

- Réalisez cette nouvelle amélioration de l'horloge.

3. Couleurs paramétrables

La dernière amélioration prévue est la possibilité de modifier les couleurs de l'horloge (couleur du cadran et couleur des aiguilles).

Ceci peut être réalisé par l'ajout de deux propriétés publiées : CouleurFond et CouleurTrait de type TColor. L'intérêt de cette solution est que ces propriétés apparaîtront dans l'inspecteur d'objet en phase de conception (elles seront également accessibles en phase d'exécution).

Le principe de réalisation d'une propriété est le suivant (exemple d'une propriété entière):

- La propriété est déclarée dans la partie publiée (published) de la classe.

```
property maPropriete : integer;
```

- Une donnée de même type est déclarée en partie privée.

```
FmaPropriete : integer;
```

- Les méthodes accès à la propriété en lecture et en écriture sont précisées dans la déclaration de la propriété. Le plus simple est un accès direct à la donnée privée.

```
property maPropriete : integer read FmaPropriete write FmaPropriete;
```

Dans ce cas, un accès en lecture comme en écriture à "maPropriete" se traduira par un accès direct à la donnée privée "FmaPropriete".

- Il est également possible d'indiquer qu'un accès en écriture fait en réalité appel à une procédure. Cette procédure est une méthode privée de la classe (en général virtuelle, ce qui permet sa redéfinition) et reçoit automatiquement en paramètre la valeur affectée à la propriété. Elle permet par exemple de réaliser un contrôle.

```
property maPropriete : integer read FmaPropriete write SetMaPropriete;
```

```
procedure MaClasse.SetMaPropriete(valeur : integer);
```

```
begin
```

```
    if valeur < 0 then valeur:=0;
```

```
    FmaPropriete:=valeur;
```

```
end;
```

- L'accès en lecture peut également faire appel à une fonction ne prenant aucun paramètre et retournant une valeur du type de la propriété.

```
property maPropriete : integer read GetMaPropriete write SetMaPropriete;
```

```
function MaClasse.GetMaPropriete : integer;
```

```
begin
```

```
    return FmaPropriete;
```

```
end;
```

Remarques :

- Il existe d'autres possibilités (propriétés énumérées, index passé en paramètre aux fonctions d'accès, ...).
- Cette implémentation de la notion de propriété permet de respecter le principe d'encapsulation.
- On voit que la mémorisation des valeurs de propriétés n'est pas obligatoirement réalisée aussi simplement qu'à l'aide d'une donnée privée.

Application à l'horloge

Nous allons définir deux propriétés : CouleurFond et CouleurTrait. L'accès en lecture sera direct, l'accès en écriture doit faire appel à une procédure. En effet, lorsque l'on change la couleur de l'horloge, celle-ci doit être redessinée. Il faut donc générer un appel à la méthode "Paint". Ceci est fait en appelant la méthode "Invalidate" héritée de TgraphicControl.

Quelques informations :

- La couleur de fond est définie par Canvas.Brush.Color.
- La couleur de trait est définie par Canvas.Pen.Color.
- La couleur du texte est définie par Canvas.Font.Color.

Travail à faire

- Ajoutez la possibilité de paramétrer les couleurs de l'horloge.

D/ Ajout d'une fonction alarme

1. Réglage de l'alarme

Nous allons faire sonner ce réveil jusqu'ici un peu inutile... L'idée est de permettre à l'utilisateur de programmer une alarme (un message) en indiquant la date et l'heure.

Pour indiquer ces informations, l'utilisateur doit disposer d'une fiche de saisie comme celle présentée ci-dessous :



D'où sort cette boîte de dialogue ?

Une fiche Delphi est un objet. Appelons celle-ci "fm_dialogue". Lorsque l'on crée cette fiche, Delphi génère automatiquement :

- Une classe Tfm_dialogue qui contient des attributs (la zone de texte pour la saisie du libellé par exemple) et des méthodes (la procédure bt_okClick qui répond à l'événement OnClick du bouton bt_ok par exemple).
- Un objet de cette classe sous la forme d'une déclaration comme : var fm_dialogue : Tfm_dialogue.

Le composant Horloge est en fait représenté par plusieurs classes (Thorloge, Taiguille, ...). Chaque classe peut contenir des attributs de tout type, et notamment de type classe (une horloge "contient" trois aiguilles par exemple).

Rien ne s'oppose donc à ce que la classe Thorloge contienne un attribut de type Tfm_dialogue.

Il faudra donc :

- Déclarer cet attribut dans la partie privée de la classe : "dialogue : Tfm_dialogue".
- Penser à créer l'objet dans le constructeur de Thorloge : "dialogue.Create(self)" // self est l'horloge elle-même, propriétaire de la boîte de dialogue).
- Penser à détruire la fiche dans le destructeur de Thorloge : "dialogue.Release" // et non dialogue.free (la méthode Release détruit la fiche après la fin d'exécution des éventuelles procédures événementielles qui lui sont liées).

Pour mémoriser l'alarme programmée, la classe Thorloge doit également disposer des attributs suivants :

- Un booléen `alarme` // est-elle programmée ?
- Un objet de type `TDateTime` `momentAlarme` // la date et l'heure de l'alarme.
- Une chaîne de caractères `libelleAlarme` // le texte du message associé à l'alarme.

Ouverture et fermeture du dialogue

L'ouverture de la boîte de dialogue sera faite sur l'événement double-clic de l'horloge elle-même. Il faut pour cela définir une méthode `horlogeDbClick` chargée d'afficher la fiche et indiquer cette méthode comme gestionnaire de l'événement double-clic sur le composant par `" OnDbClick:=horlogeDbClick;"` (cette affectation doit être faite dans le constructeur de la classe Thorloge).

La boîte de dialogue va être affichée de manière modale : l'utilisateur devra cliquer ok ou annuler avant de pouvoir continuer à faire autre chose.

L'ouverture modale est faite par la méthode `ShowModal` qui retourne une information de type `ModalResult` permettant de savoir sur quel bouton l'utilisateur a cliqué :

```
if dialogue.ShowModal=mrOk then
    // renseigner alarme, momentAlarme et libelleAlarme.
endif
```

La valeur de retour associée à un bouton peut être programmée de deux manières :

- En paramétrant la propriété `ModalResult` du bouton en phase de conception (ce sera le cas pour le bouton annuler, `ModalResult = mrCancel`).
- En affectant le résultat directement dans le code associé au bouton, ce qui permet d'effectuer d'autres traitements (ce sera le cas pour le bouton ok).

```
procedure Tfm_dialogue.bt_okClick(Sender: TObject);
begin
    // traitement de contrôle de la date et de l'heure d'alarme
    ModalResult:=mrOk;
end;
```

Saisie de la date et de l'heure

Les zones de saisie date et heure sont des composants `TDateTimePicker` (respectivement `ntp_date` et `ntp_heure`). Ce composant est paramétrable et peut donc représenter une date (`TDate`) ou une heure (`TTime`).

Pour obtenir une valeur de type `DateHeure` (`TDateTime`), il suffit en fait d'ajouter une valeur de type date et une valeur de type heure. En effet, une valeur de type `TDateTime` est un réel dont la partie entière représente la date et la partie décimale représente l'heure. Il faudra donc affecter la valeur `Time` du composant `dtp_heure` à la propriété `Time` du composant `dtp_date`. A l'issue de cette affectation, la propriété `"DateTime"` du composant `"dtp_date"` contiendra le moment complet saisi par l'utilisateur.

Déclenchement de l'alarme programmée

A chaque déclenchement du composant `Timer`, il faut vérifier si l'alarme est programmée et envoyer le message prévu si le moment est venu.

Travail à faire

- Créez la fiche de paramétrage de l'alarme. Il n'est pas nécessaire qu'elle appartienne au projet de test, il suffit qu'elle soit mentionnée dans la clause `uses` de l'horloge. Par contre, elle devra être intégrée au paquet contenant le composant.
- Modifiez la classe Thorloge en y ajoutant les attributs permettant de mémoriser les données de l'alarme et la boîte de dialogue.
- Modifiez le constructeur et le destructeur de l'horloge pour prendre en compte le dialogue.

- Ajoutez une gestion de l'événement double-clic sur l'horloge.
- Modifiez la procédure événementielle déclenchée par le timer.
- Testez le tout.

2. Réaction paramétrable

L'utilisateur de notre composant peut avoir envie de déclencher un traitement particulier lors du déclenchement de l'alarme (au lieu d'afficher un simple message) : jouer un concerto de Mozart, formater le disque dur, etc...

Afin de lui offrir cette possibilité, il faut publier la procédure à déclencher comme une propriété de l'horloge. Cette propriété est de type "procedure of object", c'est-à-dire qu'elle représente l'adresse d'une méthode. Le programmeur pourra donc y mettre n'importe quelle fonction membre de la classe du formulaire principal de son application.

Pour pouvoir déclarer une telle propriété, il faut créer un type au début du fichier horloge.pas :

```
type TprocAlarme=procedure of object;
```

Il suffit ensuite de :

- Déclarer une propriété publiée OnAlarme de type TprocAlarme.
- De lui associer une donnée privée FonAlarme de type TprocAlarme (en lecture comme en écriture).
- De créer une méthode "par défaut" de gestion de l'alarme et de l'affecter à la propriété OnAlarme à la création de l'horloge.
- De fournir une méthode publique retournant le libellé de l'alarme. Ce libellé pourrait en effet être utile au programmeur souhaitant programmer lui-même la procédure d'alarme.
- D'appeler la méthode pointée par la propriété OnAlarme lors du déclenchement. Ceci peut être fait par le code ci-dessous après vérification de l'existence d'une méthode :

```
if Assigned(OnAlarme) then OnAlarme;
```

Travail à faire

- Réalisez les modifications nécessaires.
- Testez le composant sans l'installer.
- Installez votre composant.
- Créez une application qui l'utilise et définit une méthode particulière à appeler en cas d'alarme (la propriété OnAlarme apparaît dans les événements liées à l'objet horloge).