Introduction à l'Interaction Homme Machine

Gaëtan Rey
Gaetan.Rey@unice.fr

DUT Informatique – Mars 2016

Malheureusement aujourd'hui

- Trop de systèmes sont inadaptés

C'est quoi l'IHM ?

- Interface / interaction Homme-Machine

ETUDE DE PHÉNOMÈNES MIS EN JEU DANS L'ACCOMPLISSEMENT DE TÂCHES AVEC UN SYSTÈME INFORMATIQUE

- Quels types de phénomènes ?
  - cognitifs
  - matériels
  - logiciels
  - sociaux

Objectifs du domaine de l'IHM

- Spécifier, concevoir et développer des systèmes, dispositifs, outils, machine...

Utiles
En conformité avec les fonctions attendues par l'utilisateur cible

Désirables
En conformité avec les valeurs de l'utilisateur cible

Fonctionnalité

Utilisateur

Plateforme

Environnement

Utilisables
En conformité avec les capacités cognitives, sensori-motrices de l'utilisateur cible : confort, efficacité, sécurité, qualité du produit de la tâche réalisée avec le système

Contextualisé
En conformité avec le contexte d'interaction :
- plate-forme d'interaction
- environnement physique et social

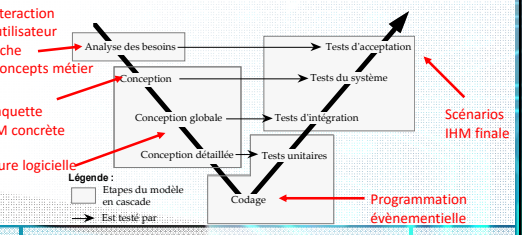
Objectifs du cours

- De manière générale, à la fin de ce module, chaque étudiant devra avoir compris comment spécifier, concevoir et développer les interfaces/interactions avec l'utilisateur.
- C'est à dire que chaque étudiants devra être capable :
  - d'identifier et de nommer (en français et en anglais) les différents composants constituant une interface graphique
  - de spécifier une interface graphique, c'est à dire :
    - de décrire l'utilisateur type à l'aide du modèle de Rasmussen
    - de construire l'arbre des tâches de l'application selon la notation HTA (Hierarchical Task Analysis)
    - d'utiliser l'approche par scénario pour guider son analyse des besoins
  - de concevoir une interface graphique, c'est à dire :
    - de juger une interface à l'aide des enseignements du modèle du processeur humain, des leçons du modèle d'ICS et de la théorie de l'action
      - de choisir les composants d'interface qui favorisent l'affordance
      - de comparer des interfaces en fonction de la loi de fitts
      - d'organiser des composants selon les règles de groupage
    - de produire des maquettes basses et hautes fidélités d'une interface à l'aide d'outils de prototypage (mockup)
    - d'argumenter les choix faits lors de la conception des maquettes en fonction des propriétés ergonomiques
  - de développer une interface graphique, c'est à dire :
    - de produire une interface à l'aide du langage de programmation Java
      - d'organiser l'architecture de l'application (les classes Java) en fonction du principe de séparation des préoccupations
    - de mettre en œuvre le patron de conception observateur/observable pour les interactions entre l'utilisateur et l'interface
      - d'utiliser la programmation évènementielle en java

IHM et Génie Logiciel

- L'IHM s'inscrit dans un processus de développement du Génie Logiciel

Contexte d'interaction
Modèle de l'utilisateur
Modèle de tâche
Modèle des concepts métier

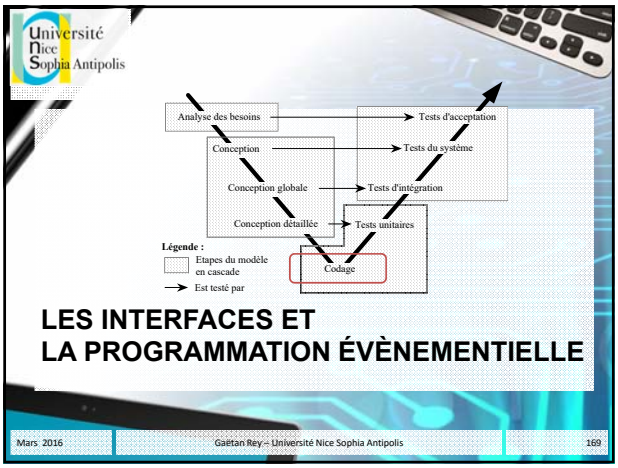


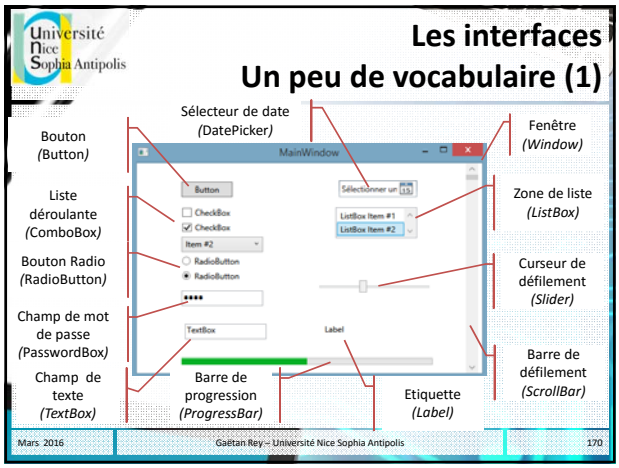
Maquette IHM concrète

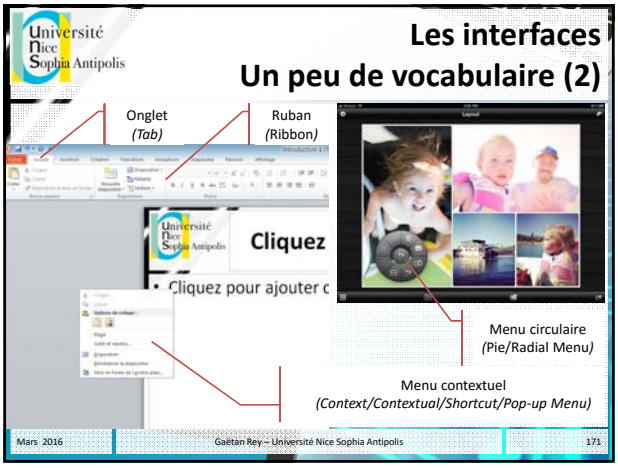
Architecture logicielle

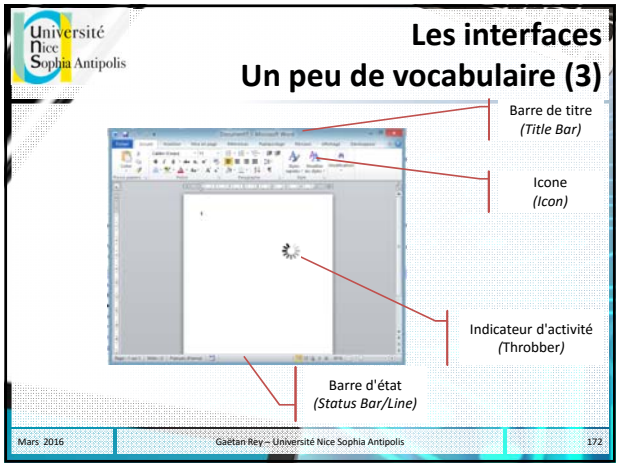
Scénarios IHM finale

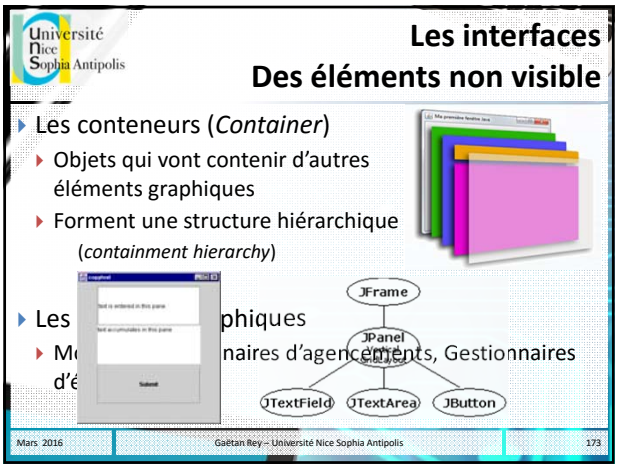
Programmation évènementielle

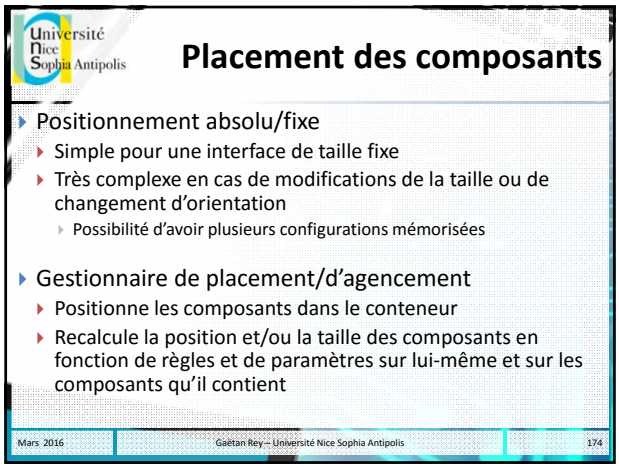

Université Nice Sophia Antipolis
Analyse des besoins
Tests d'acceptation
Conception
Tests du système
Conception globale
Tests d'intégration
Conception détaillée
Tests unitaires
Codage
Légende :
Etapes du modèle en cascade
Est testé par
LES INTERFACES ET
LA PROGRAMMATION ÉVÈNEMENTIELLE
Mars 2016
Gaëtan Rey – Université Nice Sophia Antipolis
169


Université Nice Sophia Antipolis
Les interfaces
Un peu de vocabulaire (1)
Sélecteur de date (DatePicker)
Bouton (Button)
Fenêtre (Window)
Liste déroulante (ComboBox)
Zone de liste (ListBox)
Bouton Radio (RadioButton)
Curseur de défilement (Slider)
Champ de mot de passe (PasswordBox)
Barre de défilement (ScrollBar)
Champ de texte (TextBox)
Barre de progression (ProgressBar)
Etiquette (Label)
Mars 2016
Gaëtan Rey – Université Nice Sophia Antipolis
170


Université Nice Sophia Antipolis
Les interfaces
Un peu de vocabulaire (2)
Onglet (Tab)
Ruban (Ribbon)
Cliquez
Menu circulaire (Pie/Radial Menu)
Menu contextuel (Context/Contextual/Shortcut/Pop-up Menu)
Mars 2016
Gaëtan Rey – Université Nice Sophia Antipolis
171


Université Nice Sophia Antipolis
Les interfaces
Un peu de vocabulaire (3)
Barre de titre (Title Bar)
Icone (Icon)
Indicateur d'activité (Throbber)
Barre d'état (Status Bar/Line)
Mars 2016
Gaëtan Rey – Université Nice Sophia Antipolis
172


Université Nice Sophia Antipolis
Les interfaces
Des éléments non visible
Les conteneurs (Container)
Objets qui vont contenir d'autres éléments graphiques
Forment une structure hiérarchique (containment hierarchy)
JFrame
JPanel
JTextField
JTextArea
JButton
Mars 2016
Gaëtan Rey – Université Nice Sophia Antipolis
173


Université Nice Sophia Antipolis
Placement des composants
Positionnement absolu/fixe
Simple pour une interface de taille fixe
Très complexe en cas de modifications de la taille ou de changement d'orientation
Possibilité d'avoir plusieurs configurations mémorisées
Gestionnaire de placement/d'agencement
Positionne les composants dans le conteneur
Recalcule la position et/ou la taille des composants en fonction de règles et de paramètres sur lui-même et sur les composants qu'il contient
Mars 2016
Gaëtan Rey – Université Nice Sophia Antipolis
174

Gestionnaire de d'agencement

- En .Net (WPF)
  - Ils servent également de conteneurs (double rôle)
  - Grid, Canvas, StackPanel, DockPanel, …
- En Java (en swing)
  - On parle de « Layout Manager »
  - On doit l'associer à un conteneur
  - FlowLayout, BorderLayout, GridLayout, BoxLayout, …

Exemple Java : BorderLayout

- Est le gestionnaire d'agencement par défaut d'un ContentPane (conteneur principal de la JFrame)
- Divisé en 5 zones
  - Nord, Sud, Est, Ouest, Center
  - 1 seul composant par zone
- Il consacre tout l'espace du conteneur aux composants.
- Le composant du centre dispose de la place inutilisée par les autres composants


```
this.getContentPane().add(new JButton("CENTER"),BorderLayout.CENTER);
```

- Sélectionne le conteneur de la classe courante (celle-ci est une JFrame)
- Ajoute un composant dans le conteneur
- Crée un bouton anonyme avce « CENTER » comme texte
- Indique la zone dans laquelle devra être insérer notre bouton

Exemple Java : FlowLayout

- Affiche les composants horizontalement
- Passe à la ligne suivante quand il n'y a plus de place sur la ligne

- Ne calcule la position des composants que sur un changement de taille de son conteneur
  - Possibilité de forcer un calcul avec *revalidate()*
- Chaque composant a sa taille préférée

```
bouton1.setPreferredSize(new Dimension(100,50));
```


Quelques outils de constructions d'interfaces graphiques (GUI-Builder)

- Java
  - WindowBuilder (Eclipse)
  - Swing GUI Builder (Netbeans)
  - Swing GUI Designer (IntelliJ IDEA)
  - JFormDesigner (Stand-alone, Eclipse, IntelliJ IDEA, Jbuilder, Netbeans, Jdeveloper)
  - Jigloo GUI Builder (Eclipse, WebSphere Studio)
- C++
  - Visual Studio
  - Ultimate++
  - wxDev-C++
  - Qt (Stand-alone, Eclipse, Netbeans)
- .Net
  - Visual Studio
  - SharpDevelop
  - MonoDevelop
- GTK+
  - Glade (C, C++, C#, Vala, Java, Perl, Python ...)

Programmation séquentielle

- Programmation classique / « procédurale »
  - Les instructions s'exécutent dans un ordre donné
  - Les unes après les autres
- Interactions avec l'utilisateurs
  - Au fil des instructions du code
  - L'exécution est bloquée attendant l'utilisateur

```
int nombre = 0;
int res = 0;
printf("Entrez une valeur");
res = scanf("%d", &nombre);
if (res == 1)
    printf("Votre valeur est %d", nombre);
else
  // traitement de l'erreur
```

- Le déroulement est contrôlé par une séquence d'instructions écrites

Problème du blocage

- Exemple : un lecteur audio
  - Si attendre une action de l'utilisateur bloque le programme :
    - Comment mettre en pause ?
    - Comment changer le volume pendant la lecture ?
- Avoir un mécanisme non bloquant
  - La programmation évènementielle

Programmation évènementielle

- Paradigme de programmation fondé sur les événements
- Le déroulement est contrôlé par la survenue d'événements (dont les actions de l'utilisateur)
- Principe
  - On s'abonne à des évènements
    - auprès une certaine ressource (Observable)
  - Quand l'évènement ce produit
    - les abonnées (Observateur) sont avertis

Evènements (1)

- Un événement
  - un message transmis par un sujet (objet)
  - pour signaler l'occurrence d'une action ou d'un changement d'état
- L'action peut être
  - provoquée par l'intervention de l'utilisateur,
    - Exemple : un clic de bouton, …
  - levée par une autre logique de programme
    - Exemple : modifier une valeur d'une propriété
- Le sujet (objet) qui déclenche l'événement est appelé *l'émetteur d'événements*
- L'émetteur d'événement ne sait pas quel objet ou méthode va recevoir (gérer) les événements qu'il lève.

Evènements (2)

- Un processus en 4 étapes
  - **S'abonner** : indiquer à une entité qu'on est intéressé par telle information
  - **Attendre** : rien à faire
  - **Être notifié** : l'entité nous informe qu'il vient de se passer quelques chose
    - C'est la réception de l'évènement
  - **Réagir** : faire une action en fonction de l'information associée à l'évènement

Comparaison

Programmation séquentielle/classique

- Le programmeur écrit la boucle principale
- Programme

```
point d'entrée
initialisations
répéter
   lire une commande
   traiter une commande
jusqu'à la commande finir
```

Programmation évènementielle

- Pas de boucle principale
  - elle est enfouie dans la bibliothèque
  - L'ordre d'exécution dépend des évènements, il n'est pas visible dans le programme
- Programme

```
fonctions (réaction aux événements)
point d'entrée
initialisations
initialisation // enfouie dans la bibliothèque
tant que (non fin) {
  attendre évènement suivant E
  traiter évènement E
}
```

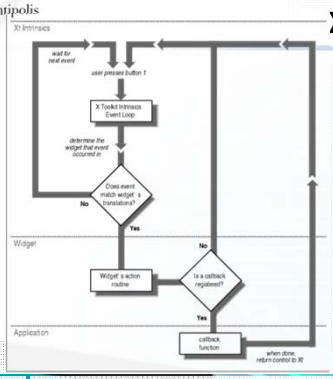

Programmation événementielle

- Comment cela fonctionne t il?
  - Les événements sont placés dans une liste FIFO
  - La boucle de gestion des événements prend les événements dans la liste et les traite
    - XtAppMainLoop() en c (Xt + Motif)
    - gtk_main() en GTK+
    - Event Dispatch Thread en Java
    - UI thread en C#
    - …

XtAppMainLoop() Xt + Motif

- La gestion des évènements est prise en charge par XT
- La fonction XtAppMainLoop :
  - lit le prochain évènement : XtAppNextEvent
  - l'envoie à la procédure appropriée: XtDispatchEvent
- Cette fonction utilise le champ window de l'évènement pour chercher une widget qui possède cette fenêtre
  - Les clients qui ont sélectionné les bons évènements sur les (fenêtres des) widgets les reçoivent, et les traitent
  - Cette gestion est "automatique" pour les widgets qui ont des *tables de correspondances*
  - Elle est à un niveau d'abstraction supérieur à la gestion des évènements bruts

XtAppMainLoop()
Xt + Motif (2)



[The Motif Programming Model]

Mars 2016 Gaëtan Rey – Université Nice Sophia Antipolis 187

gtk_main()
GTK+

- La fonction gtk_main()
  - Se « connecte » au serveur X
  - Utilise une file d'attente d'évènements
- gtk_main()
  - attendra les événements venant du serveur X et
  - demandera aux widgets d'émettre les signaux lorsque ces événements surviendront.

Mars 2016 Gaëtan Rey – Université Nice Sophia Antipolis 188

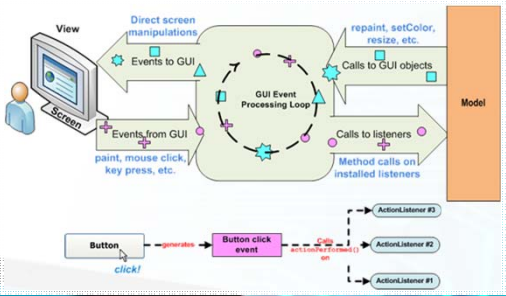
Event Dispatch Thread
Java

- EDT est la Thread dans laquelle l'AWT et Swing font
  - leur affichage
  - leur propagation d'événements.
- Les composants AWT et Swing doivent uniquement être utilisés dans l'EDT
- L'EDT alterne
  - les cycles d'affichage
  - les cycles de propagation d'événements.
- Lors de la gestion des évènements ou des composants graphiques, il faut vérifier:
  - Qu'on est dans l'EDT
    - AWT : invoquer la méthode statique isDispatchThread() de la classe java.awt.EventQueue.
    - Swing : invoquer la méthode statique isEventDispatchThread() de la classe javax.swing.SwingUtilities
  - Que le composant n'est pas en train de se redessiner

Mars 2016 Gaëtan Rey – Université Nice Sophia Antipolis 189

Event Dispatch Thread
Java

Source : https://www.clear.rice.edu/comp310/JavaResources/GUI/



Mars 2016 Gaëtan Rey – Université Nice Sophia Antipolis 190

Thread d'interface utilisateur
C#

- Les applications WPF commencent avec 2 threads
  - un pour gérer le rendu qui s'exécute masqué en arrière-plan (rendering thread)
  - un pour gérer l'interface utilisateur qui reçoit les entrées, gère les événements, peint l'écran et exécute le code de l'application (UI thread)
- Le thread d'interface utilisateur met en file d'attente des éléments de travail à l'intérieur d'un objet appelé un Dispatcher
- Le Dispatcher sélectionne les éléments de travail en fonction de leur priorité et exécute complètement chacun d'eux

Mars 2016 Gaëtan Rey – Université Nice Sophia Antipolis 191

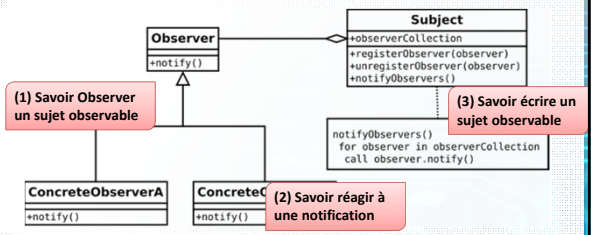
Patron de conception
Observateur/Observable

- Appelé également Observer / Observable
- Met en œuvre plusieurs acteurs
  - le sujet (observable) et les observateurs
- Permet à un ensemble d'objets
  - de s'abonner à un sujet dit observable
  - pour recevoir des notifications quand ce sujet change
- Objectif
  - augmenter la réutilisabilité en diminuant le couplage entre les classes
    - le widget bouton peut être utilisé dans plusieurs projets. Il n'y a pas de lien entre le widget et les fonctions appelées dans chaque projet

Mars 2016 Gaëtan Rey – Université Nice Sophia Antipolis 192

Patron de conception Observateur/Observable

- La notion d'**observateur/observable** permet de coupler des modules de façon à réduire les dépendances aux seuls phénomènes observés



[Wikipédia]

Abonnement et réaction en C (Xt + Motif) (1)

- L'émission d'un évènement va entrainer l'exécution d'une ou plusieurs fonctions
  - Les « callback »
- Les fonctions de callback possèdent trois paramètres :
  - Un pointeur sur le composant (Widget) associé à ce callback.
  - Le second paramètre sert à passer des données à la fonction de callback
  - Un pointeur sur une structure qui contient
    - tous les renseignements relatifs à l'événement qui a provoqué l'appel de la fonction de callback,
    - des renseignements relatifs au type de Widget passé en 1er paramètre,

Abonnement et réaction en C (Xt + Motif) (2)

```
// Définie une fonction de callback
void MaFonction(Widget w, XtPointer cld, XtPointer cad) {
  // Mettre ici le code de la fonction
}

int main() {
   Widget boite, bouton;
    …
   // Crée un bouton avec boite comme parent
   bouton = XmCreatePushButton(boite, "bouton", NULL, 0);
   // Rend le bouton visible
   XtManageChild(bouton);
   // Associe un callback -> appelez MaFonction quand on « active » le bouton
   XtAddCallback(bouton, XmNactivateCallback, MaFonction, NULL);
   // lance la boucle de gestion des événements
   XtAppMainLoop(app);
   // Attention, rien n'est exécuté après cette ligne !
}
```


Abonnement et réaction en C (GTK+) (1)

- En GTK+, un signal est une méthode qui permet de recevoir de notification
  - un signal est un événement qui se produit sur un objet
- Ce sont les signaux qui permettent à l'utilisateur d'une application d'interagir avec l'application.
  - Par exemple, si un utilisateur clique sur un bouton, un signal d'un objet sera émis dans l'objet spécifié.
- La fonction spécifiée lors de l'émission de ce signal, sera appelée
- Un signal est sélectionné en utilisant un identificateur entier (l'id du signal) ou une chaîne de caractères
- La fonction qui sera appelée lors de l'émission du signal, est appelée fonction callback

[Cours GTK 2]
[Les signaux]

Abonnement et réaction en C (GTK+) (2)

```
void OnDestroy(GtkWidget *pWidget, gpointer pData){
  /* Arrêt de la boucle evenementielle */
  gtk_main_quit();
}
int main(int argc,char **argv){
   /* Déclaration du widget */
   GtkWidget *pWindow;
   gtk_init(&argc,&argv);
   /* Création de la fenêtre */
   pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
   /* Connexion du signal "destroy" */
   g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(OnDestroy), NULL);
   /* Affichage de la fenêtre */
   gtk_widget_show(pWindow);
   /* Demarrage de la boucle evenementielle */
   gtk_main();
   return EXIT_SUCCESS;
}
```


Abonnement et réaction en Java (1)

- Dans le contexte d'une interface graphique (Swing, AWT, …), les listeners permettent au programmeur de réagir suite aux actions de l'utilisateur
- Les « listeners » sont des interfaces
  - fournissent une ou plusieurs méthodes qui peuvent être implémentées différemment selon les cas et les besoins, pour répondre aux événements
- Chaque listener dispose d'une classe Event associée. Cette classe
  - étend java.awt.event.EventObject
  - fournit une description de l'évènement capturé.

[Developpez.com]

Abonnement et réaction en Java (2)

```
// Étape 1 : déclaration de la classe
public class MaClasse {
        // Étape 2 : création d'un bouton.
        JButton monBouton = new JButton("Mon Bouton...");

        /* Étape 3 : création de la  classe anonyme */
        ActionListener listener = new ActionListener() {
                public void actionPerformed(ActionEvent e) {

                        // Bouton a été cliqué

                }
        };
        monBouton.addActionListener(listener);

}
```

[Developpez.com]

Abonnement et réaction en Java (3)

```
// Étape 1 : déclaration de la classe
public class MaClasse {
        // Étape 2 : création de deux boutons.
        JButton monBouton = new JButton("Mon Bouton...");
        JButton monBouton2 = new JButton("Mon Bouton 2...");
        /* Étape 3 : création de la  classe anonyme */
        monBouton.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                // Cette méthode ne sera appelée que pour les évènements sur le bouton monBouton.
                }
        });
        /* On refait la même chose pour le deuxième bouton */
        monBouton2.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                // Cette méthode ne sera appelée que pour les évènements sur le bouton monBouton2.
                }
        });
}
```

[Developpez.com]

Abonnement et réaction en Java (4)

```
// Étape 1 : déclaration de la classe
public class MaClasse implements ActionListener {
        // Étape 2 : Création de deux boutons
        JButton monBouton = new JButton("Mon Bouton");
        JButton monBouton2 = new JButton("Mon Bouton2");

        public MaClasse() {
                // Étape 4 : On ajoute « l'écouteur » sur le bouton « monBouton »
                monBouton.addActionListener(this);
                // Puis sur monBouton2
                monBouton2.addActionListener(this);
        }

        /* Étape 3 :Cette méthode est déclarée dans l'interface ActionListener. Il nous faut l'implémenter. */
        public void actionPerformed(ActionEvent e) {
                // Étape 2bis
                if(e.getSource() == monBouton) {
                        // Bouton 1 a été cliqué
                }else {
                        // Bouton 2 a été cliqué
                }
        }
}
```

[Developpez.com]

Abonnement et réaction en Java (5)

```
// Fichier : MonListener.java
public class MonListener implements ActionListener{
        public void actionPerformed(ActionEvent e) {
                if(e.getSource() == monBouton) {
                        // Bouton 1 a été cliqué
                }else {
                        // Bouton 2 a été cliqué
                }
        }
}
// Fichier : MaClasse .java
public class MaClasse {
        JButton monBouton = new JButton("Mon Bouton");
        JButton monBouton2 = new JButton("Mon Bouton2");

        public MaClasse() {
                monBouton.addActionListener(new MonListener ());
                monBouton2.addActionListener(new MonListener ());
        }

}
```

[Developpez.com]

Abonnement et réaction en C# (1)

- Utilisation de délégués pour la gestion d'événements
- Un délégué
  - Est un type qui encapsule (contient une référence à) une méthode
  - Est déclaré avec une signature
    - qui indique le type de retour et les paramètres des méthodes qu'il référence,
    - et peut contenir des références à des méthodes qui correspondent à la signature.
  - Est l'équivalent d'un appel de fonction ou d'un pointeur de fonction de type sécurisé.
  - Une déclaration delegate suffit pour définir une classe déléguée
  - Une fois initialisé avec une méthode, il se comporte exactement comme cette méthode et peut être appelé avec l'opérateur ()

```
public delegate void SeuilAtteintEventHandler(SeuilAtteint EventArgs e);
```


Abonnement et réaction en C# (2)

```
// On indique le type de delegate défini pour l'évènement (EventHandler )
monObjet.MonEvenement += new EventHandler(monObjet_MonEvenement);
/*  La signature (type de retour et types des paramètres) de la méthode utilisée pour
s'abonner doit correspondre à la signature de ce delegate */
private void monObjet_MonEvenement(object sender, EventArgs e)
{
  Console.WriteLine("MonEvenement s'est produit !");
}

// Depuis C# 2, il n'est plus nécessaire de spécifier le type de delegate
monObjet.MonEvenement += monObjet_MonEvenement;

// Pour se désabonner d'un évènement
monObjet.MonEvenement -= monObjet_MonEvenement;
```

Problème du Multi-thread

- Que faire
  - Si j'ai un traitement de mon évènement qui est trop long et qui bloque ma GUI ?
    - Utiliser un autre thread !
  - Si je veux modifier ma GUI depuis un autre thread ?
    - On ne peut pas !
    - Mais on peut contourner le problème
      - Demander au thread qui gère la GUI de faire le travail
        - invokeLater en Java
        - Invoke ou BeginInvoke en C#

Gestion de XtAppMainLoop() en Xt + Motif

- Motif est étroitement couplé avec la X Toolkit
- Les interfaces X11R6 pour le multi-threading ne sont nécessaires que si l'application multithread appelle les interfaces Motif / Xt dans plusieurs threads
- Il est possible d'écrire des applications multi-thread Motif dans lesquelles les interfaces Motif / Xt sont invoquées qu'à partir d'un seul fil.
  - Dans de telles applications, les interfaces ne sont pas nécessaire

[Structure of a Motif Program]

Gestion de gtk_main() en GTK+

- Une réponse simple
  - Dans le cas d'une gestion multi-thread, vous ne devez appeler la bibliothèque GTK seulement depuis le thread exécutant gtk_main()
- En réalité, c'est plus compliqué
  - [Multi-threaded GTK applications – Part 1: Misconceptions]
  - [Multi-threaded GTK applications – Part 2: java-gnome]

Gestion de l'EDT en Java

- Exécuter un traitement plus tard dans l'EDT (ou depuis un autre Thread)

```
// Cette méthode retourne immédiatement.
SwingUtilities.invokeLater(new Runnable() {
    /**
     * {@inheritDoc}
     */
    @Override public void run() {
        myButton.setText("Salut le monde !") ;
    }
}) ;
```

- Lancer un traitement long
  - Doivent être lancés hors de l'EDT
  - Pour Java 6+, utilisez la classe javax.swing.SwingWorker qui offre un framework permettant d'exécuter une tache hors de l'EDT tout en récupérant ses résultats intermédiaires et finaux dans l'EDT.

Gestion de l'UI Thread en C# (1)

```
// La mauvaise façon de faire

// Travail effectué depuis un autre thread
ThreadStart start = delegate() {
// ...

// Ceci provoquera une exception
statusText.Text = "Depuis un autre Thread";
};

// Crée et démarre un thread
new Thread(start).Start();
```

```
// La bonne façon de faire (appel synchrone)

// Travail effectué depuis un autre thread
ThreadStart start = delegate() {
// ...

// Met à jour le Texte du TextBlock.
// Cette action sera effectuée via le dispatcher
Dispatcher.Invoke(DispatcherPriority.Normal,
                  new Action<string>(SetStatus),
                  "Depuis un autre Thread ");
};
// Crée et démarre un thread
new Thread(start).Start();
```

[Créez des applications plus réactives avec le répartiteur]

Gestion de l'UI Thread en C# (2)

```
// La bonne façon de faire (appel asynchrone)
// Travail effectué depuis un autre thread
ThreadStart start = delegate() {
// ...
        // Met à jour le Texte du TextBlock.
        // Cette action sera effectuée via le dispatcher
        DispatcherOperation op = Dispatcher.BeginInvoke( DispatcherPriority.Normal, new
                        Action<string>(SetStatus), " Depuis un autre Thread (Async)");
        DispatcherOperationStatus status = op.Status;
        while (status != DispatcherOperationStatus.Completed) {
                status = op.Wait(TimeSpan.FromMilliseconds(1000));
                if (status == DispatcherOperationStatus.Aborted) {
                        // Alert Someone
                }
        }
};
// Crée et démarre un thread
new Thread(start).Start();
```

Nos Evènements

- Le mécanisme d'évènements n'est pas figé
- On peut créer nos propres évènements
- On peut émettre des évènements dans nos programmes

Définir des évènements en Java (1)

- Pour créer ses propre évènements
  - Une classe évènement

```
public class MyEventClass extends java.util.EventObject {
        //le constructeur de la class
        public MyEventClass(Object source) {
          super(source);
        }
}
```

  - Une Interface

```
public interface MyEventClassListener {
      public void handleMyEventClassEvent(EventObject e);
}
```


Définir des évènements en Java (2)

- Pour créer ses propre évènements
  - Une source de l'évènement

```
public class MyEventSource {
      private List _listeners = new ArrayList();
      public synchronized void addEventListener(MyEventClassListener listener)  {
            _listeners.add(listener);
      }
      public synchronized void removeEventListener(MyEventClassListener listener)  {
            _listeners.remove(listener);
      }
      // call this method whenever you want to notify the event listeners of the particular event
     private synchronized void fireEvent() {
            MyEventClass event = new MyEventClass(this);
            Iterator i = _listeners.iterator();
            while(i.hasNext())  { ((MyEventClassListener) i.next()).handleMyEventClassEvent(event);}
      }
```


Définir des évènements en Java (3)

- Pour créer ses propre évènements
  - Une Event Source

```
public class MyEventListener implements MyEventClassListener {
        // ... Votre code

        //implement la(les) méthode(s) requise de l'interface
        public void handleMyEventClassEvent(EventObject e)   {
              // gérer l'événement comme bon vous semble
        }
}
```

[Creating a Custom Event]

Définir des évènements en C# (1)

- Définir d'un évènement

```
class MyClass{
        public event EventHandler MyEvent;
}
```

  - Utilisez event dans la signature de votre classe d'événements
  - Spécifiez le type du délégué pour l'événement

[Accesseurs d'événement]

Déclencher des évènements en C# (2)

- Déclencher un événement
  - Ajoutez une méthode marquée comme protected et virtual
  - Nommez cette méthode OnEventName / QuandNomEvenement
  - La méthode prend un paramètre de type « objet de données d'événement »

```
class MyClass {
        public event EventHandler MyEvent;

        protected virtual void OnMyEvent(EventArgs e) {
              EventHandler handler = MyEvent;
              if (handler != null) {
                    handler(this, e);
              }
        }
        // le reste du code
}
```

Le minimum à connaître

- Noms des différents Widgets
- Notions de conteneurs et de gestionnaire d'agencement
- Principe de le programmation évènementielle
- Notion d'évènements
- Patron de conception « observer/observable »
- Notion de Callback, listerner, delegate

Bibliographie / Remerciement

- Interaction Homme-Ordinateur de J. Coutaz
- Human-Computeur Interaction 2de Edition de A. Dix, J. Finlay, G. Abowd et R. Beale
- The Human-Computer Interaction Handbook edité par J. A. Jacko et A. Sears

- Ce cours a été construit à l'aide des supports de
  - Philippe Renevier
  - Jerôme Henrique
  - Joëlle Coutaz
  - Fabien Duchateau et Stéphanie Jean-Daubias
- Un grand merci à eux !!