

TP/TD

#01 Frame Version 01

```

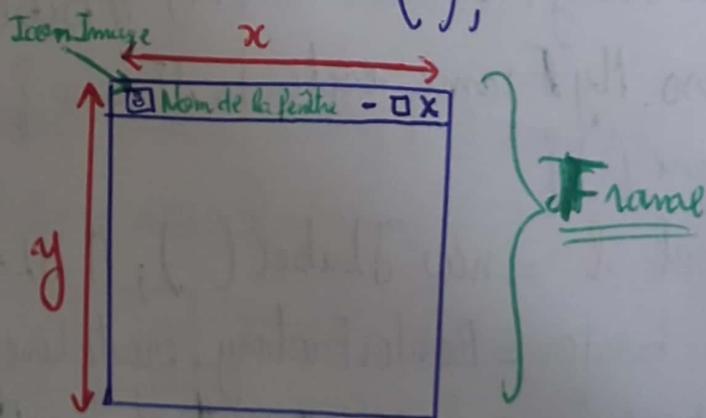
import javax.swing.*; ← Bibliothèque
public class NameClass extends JFrame {
    public NameClass(){} ↑ Le nom de la classe
    this.setTitle ("Nom de la fenêtre"); // Donne un nom pour la fenêtre
    this.setSize (420, 400); // Donne la taille de la fenêtre (x, y)
    this.setDefaultCloseOperation (EXIT_ON_CLOSE); // Exit out of
the application */
    this.setLocation (500, 250); // La position de la fenêtre dans l'écran.
    this.setVisible (true); // make this visible window.
    ImageIcon image = new ImageIcon (); // create an ImageIcon
    this.setIconImage (image.getImage ()); // change icon of "this".
}

```

```
public static void main (String [] args) {
```

```
NameClass exo = new NameClass ();
```

```
}
```



Version 028

```
public NameClass {
    public static void main(String[] args) {
        JFrame frame = new JFrame(); // create frame.
        frame.setTitle("Nom de la fenêtre");
        frame.setSize(400, 250);
        frame.DefaultCloseOperation EXIT-ON-CLOSE;
        frame.setVisible(true);
    }
}
```

#02 Label & JLabel = a GUI display area for a string of text, an image, or both.

```
import javax.swing.*;
import java.awt.Color; // Bibliothèque "Color".
import java.awt.Font; // Bibliothèque "Font".
import javax.swing.BorderFactory; // Bibliothèque "Border".
public class MyFrame extends JFrame {
    public MyFrame() {
        JLabel l = new JLabel(); // "JLabel" pour ajouter un étiquette
        Border border = BorderFactory.createLineBorder(Color.green, 3)
        // pour ajouter "border" avec la couleur verte et "width=3".
```

- l. `setLabel ("Bro, do you even Code?");` // set text of Label.
- l. `setForeground (new Color(0x00FF00));` // set font color of text.
- l. `setFont (new Font ("MV Boli", Font.PLAIN, 20));`
 // set font of text.

0x000011
0x200033
↑
la taille de text

l. `setVerticalAlignment (JLabel.BOTTOM);`
 or TOP or CENTER

l. `setHorizontalAlignment (JLabel.CENTER);`
 or LEFT or RIGHT

l. `setBackground (Color.Black);` // set Background => Black color.

l. `setBorder (border);` } l. `setOpaque (true);` } // il faut écrire les deux.

this. `setTitle ("Label");`

this. `setDefaultCloseOperation (EXIT_ON_CLOSE);`

this. `setSize (400, 400);`

this. `setResizable (false);` // pour garder toujours la taille de la fenêtre
 (ne change pas).

this. `setVisible (true);`

this. `add (l);` // pour ajouter Label.

} public static void main (String [] args) {

MyFrame frame = new MyFrame ();

}

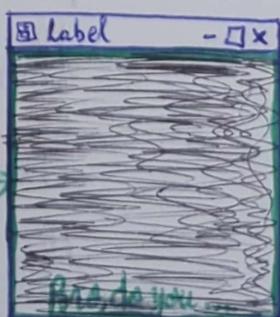
Horizontal →



Vertical

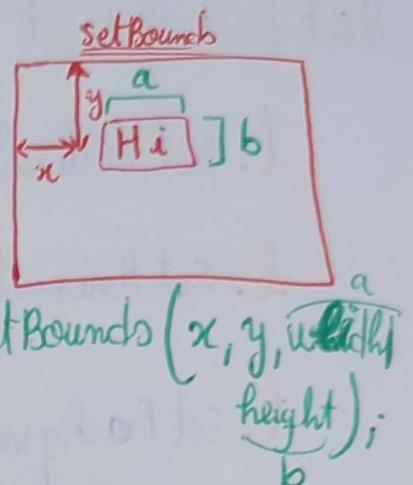
- ③ -

Border
(green)



#03 Panels

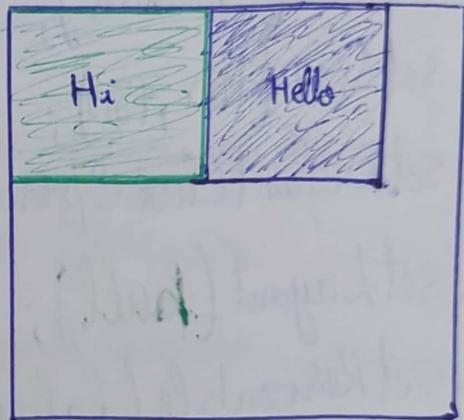
```
import java.awt.Color;  
import javax.swing.*;  
public class MyFrame extends JFrame {  
    public MyFrame() {  
        JLabel l = new JLabel("Hi");  
        JLabel lh = new JLabel("Hello");  
        l.setBounds(50, 70, 50, 50); // setBounds(x, y, width)  
        lh.setBounds(50, 70, 50, 50);  
  
        JPanel pg = new JPanel(); // ajouter deux panels.  
        JPanel pb = new JPanel();  
        pg.setBackground(Color.green);  
        pg.setBounds(0, 0, 250, 250);  
        pg.setLayout(null);  
        pg.add(l); // ajouter le text "l" dans "panel green".  
  
        pb.setBackground(Color.blue);  
        pb.setBounds(250, 0, 250, 250);  
        pb.setLayout(null);  
        pb.add(lh); // ajouter le text "lh" dans "panel blue".  
        this.setTitle("Panel");  
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);  
        this.setSize(750, 750);
```



```

        this.setResizable(false);
        this.setLayout(null);
        this.add(pg); // ajouter "panel green" dans Frame.
        this.add(pb); // ajouter "panel blue" dans Frame.
    }
    public static void main(String[] args) {
        MyFrame frame = new MyFrame();
        frame.setVisible(true);
    }
}

```



#04 Button

```

import java.awt.Color;
import javax.swing.*;
public class MyFrame extends JFrame {
    public MyFrame() {
        JLabel l = new JLabel("Button");
        l.setBounds(318, 250, 75, 75);
        JButton button = new JButton("Button");
        button.setBounds(300, 300, 75, 75);
        button.setBackground(Color.green);
    }
}

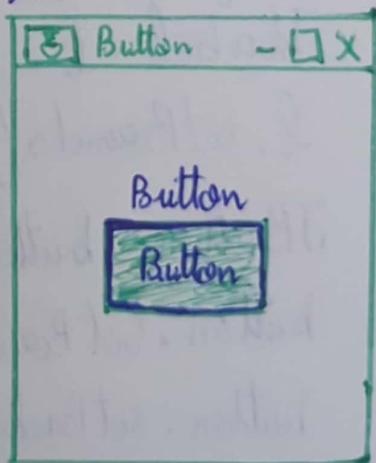
```

```

button.setBorder(BorderFactory.createLineBorder(Color.blue));
button.setFocusable(false);
button.addActionListener(e -> {
    if(e.getSource() == button) {
        System.out.println("Thanks !!");
        button.setEnabled(false); // pour une seule click de
    } // Button.
}); // }

this.setTitle("Button");
this.setSize(750,750);
this.setDefaultCloseOperation(EXIT_ON_CLOSE);
this.setLayout(null);
this.setResizable(false);
this.add(l);
this.add(button);
} // public static void main(String[] args) {
MyFrame frame = new MyFrame();
frame.setVisible(true);
}
}

```



#05 Border Layout

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

```
import javax.swing.*;
```

```
public class MyFrame extends JFrame {
```

```
    public MyFrame() {
```

```
        JPanel P1 = new JPanel();
```

```
        JPanel P2 = new JPanel();
```

```
        JPanel P3 = new JPanel();
```

```
        JPanel P4 = new JPanel();
```

```
        JPanel P5 = new JPanel();
```

```
P1. setPreferredSize(new Dimension(100, 100));
```

```
P2. setPreferredSize(new Dimension(100, 100));
```

```
P3. setPreferredSize(new Dimension(100, 100));
```

```
P4. setPreferredSize(new Dimension(100, 100));
```

```
P5. setPreferredSize(new Dimension(100, 100));
```

```
P1. setBackground(Color.green);
```

```
P2. setBackground(Color.yellow);
```

```
P3. setBackground(Color.black);
```

```
P4. setBackground(Color.blue);
```

```
P5. setBackground(Color.red);
```

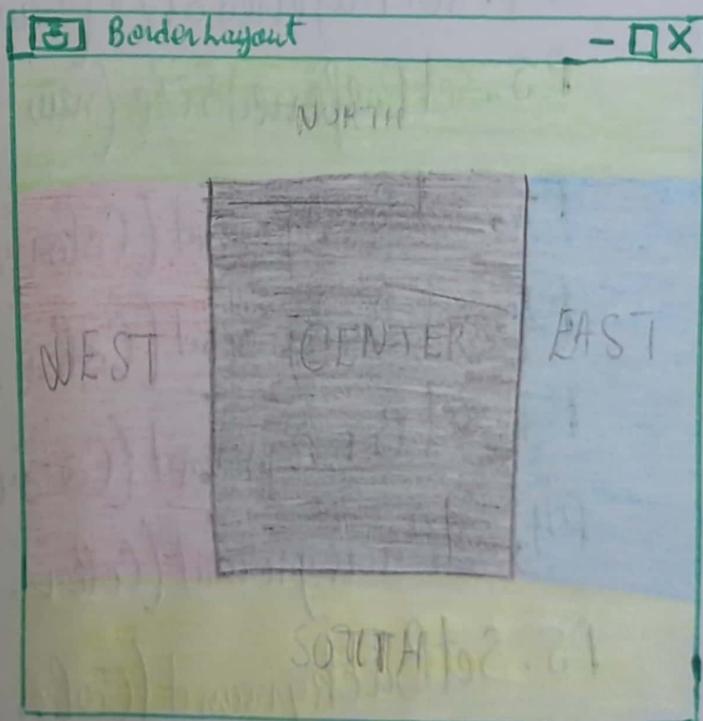
```
this.setTitle("BorderLayout");
```

```
this.setSize(500,500);  
this.setDefaultCloseOperation(EXIT_ON_CLOSE);  
this.setResizable(false);  
this.setLayout(new BorderLayout());  
this.add(P1, BorderLayout.NORTH);  
this.add(P2, BorderLayout.SOUTH);  
this.add(P3, BorderLayout.CENTER);  
this.add(P4, BorderLayout.EAST);  
this.add(P5, BorderLayout.WEST);  
}
```

```
public static void main(String[] args){
```

```
MyFrame frame = new MyFrame();
```

```
frame.setVisible(true);
```



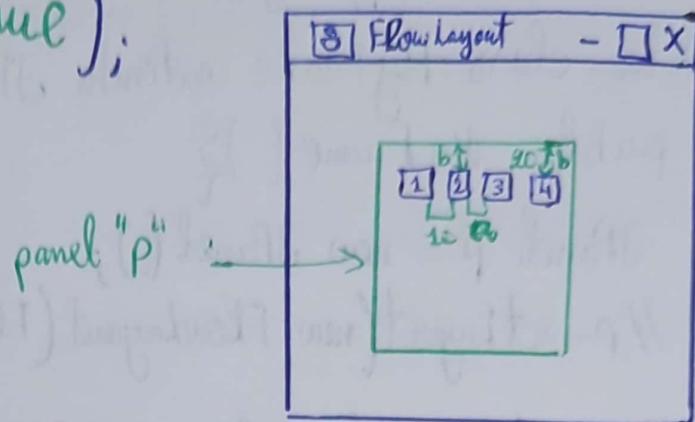
#06 FlowLayout

```
import java.awt.*;
import javax.swing.*;
public class MyFrame extends JFrame {
    public MyFrame() {
        JPanel p = new JPanel();
        // p.setLayout(new FlowLayout(FlowLayout.TRAILING, 10, 10));
        // p.setLayout(new FlowLayout(FlowLayout.LEADING, 10, 20));
        p.setLayout(new FlowLayout(FlowLayout.CENTER, 10, 20));
        p.add(new JButton("1"));
        p.add(new JButton("2"));
        p.add(new JButton("3"));
        p.add(new JButton("4"));
        p.setBackground(Color.green);
        p.setBounds(100, 100, 300, 300);
        this.setTitle("FlowLayout");
        this.setSize(500, 500);
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        this.setResizable(false);
        this.setLayout(null);
        this.add(p);
    }
}
```

```

public static void main (String [] args) {
    MyFrame frame = new MyFrame ();
    frame.setVisible (true);
}
}

```



#07 GridLayout

```

import java.awt.*;
import javax.swing.*;
public class MyFrame extends JFrame {
    public MyFrame () {
        JPanel p = new JPanel (new GridLayout (3, 3, 10, 20));
        p.add (new JButton ("1"));
        p.add (new JButton ("2"));
        p.add (new JButton ("3"));
        p.add (new JButton ("4"));
        p.add (new JButton ("5"));
        p.add (new JButton ("6"));
        p.setBackground (Color.green);
        p.setBounds (100, 100, 250, 250);
    }
}

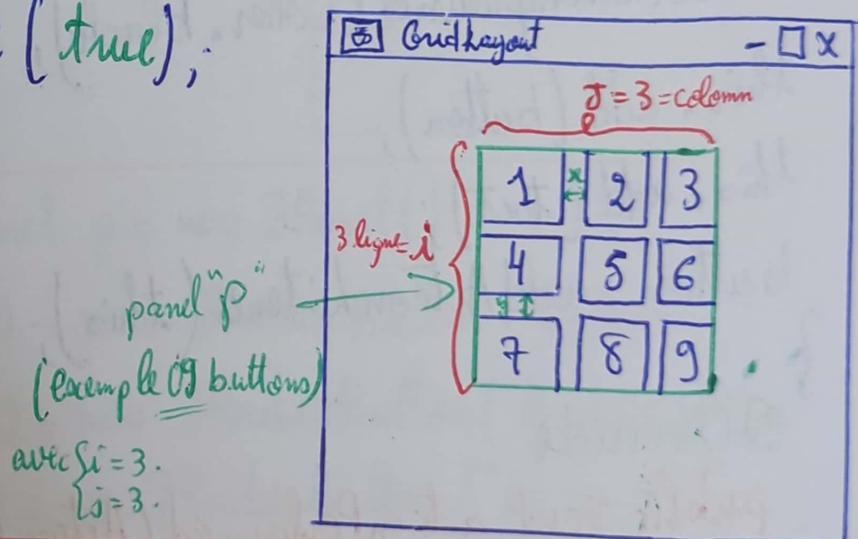
```

```

        this.setTitle("GridLayout");
        this.setSize(500, 500);
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        thisResizable(false);
        this.setLayout(null);
        this.add(p);
    }

    public static void main(String[] args) {
        MyFrame frame = new MyFrame();
        frame.setVisible(true);
    }
}

```



#08 JTextField

```

import java.awt.*;
import java.awt.event.ActionEvent;
import javax.swing.*;
import java.awt.event.ActionListener;
public class MyFrame extends JFrame implements ActionListener {
    JButton button; // Variable globale.
    JTextField tact;
    public MyFrame() {
        this.setTitle("TextField");
    }
}

```

```

this. SetDefaultCloseOperation ( EXIT_ON_CLOSE );
this. setSize ( 400, 100 );
this. setResizable ( false );
this. setLayout ( new FlowLayout ( ) );
button = new JButton ( "Submit" );
txt = new JTextField ( "Test" );
txt. setPreferredSize ( new Dimension ( 300, 40 ) );
txt. setForeground ( Color. green );
txt. setFont ( new Font ( "Time new roman", Font. PLAIN, 30 ) );
txt. setBackground ( Color. black );
this. add ( button );
this. add ( txt );
button. addActionListener ( this );
}

```

@Override

```

public void actionPerformed ( ActionEvent e ) {

```

```

if ( e. getSource ( ) == button ) {

```

```

System. out. println ( "TextField " + txt. getText ( ) );

```

```

button. setEnabled ( false );
}
}

```

```

public static void main ( String [ ] args ) {

```

```

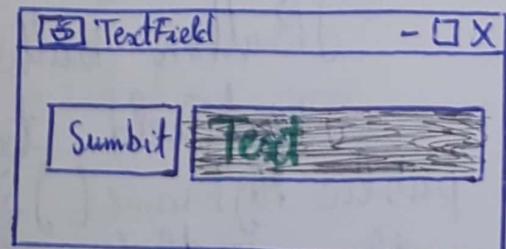
MyFrame frame = new MyFrame ( );

```

```

frame. setVisible ( true );
}
}

```



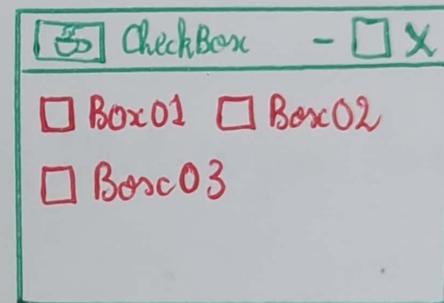
#09 CheckBox & JPanel

```

 JPanel p = new JPanel();
 JCheckBox box01 = new JCheckBox("Box01");
 JCheckBox box02 = new JCheckBox("Box02");
 JCheckBox box03 = new JCheckBox("Box03");
 box01.setMnemonic('P'); // Raccourci clavier (alt + 'P') pour sélectionner
 box02.setMnemonic('R');
 box03.setMnemonic('D');

 p.add(box01); // Pour ajouter check Box : Box01.
 p.add(box02);
 p.add(box03);
 this.add(p);

```



#10 RadioButton & JPanel

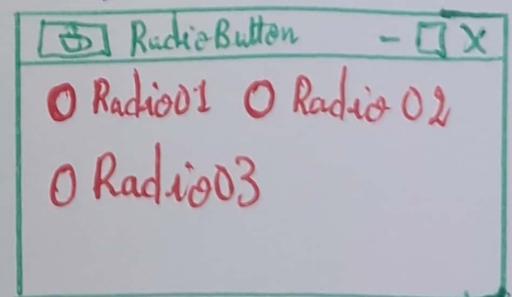
```

 JPanel p = new JPanel();
 JRadioButton radio01 = new JRadioButton("Radio01");
 JRadioButton radio02 = new JRadioButton("Radio02");
 JRadioButton radio03 = new JRadioButton("Radio03");

 p.add(radio01);
 p.add(radio02);
 p.add(radio03);

 ButtonGroup groupe = new ButtonGroup();
 groupe.add(radio01); // pour limiter la sélection .
 groupe.add(radio02);
 groupe.add(radio03);
 this.add(p);

```



IHM

Chapitre I : Notions d'interaction

1. Définition : IHM signifie "interface homme-machiné", c'est la façon dont les utilisateurs interagissent avec un ordinateur ou un système informatique. Cela peut inclure des interfaces utilisateur graphique (GUI).

Autrement dit que IHM est une interface utilisateur permettant de connecter une personne à une machine, à un système ou à un appareil.

2. L'utilisabilité d'une IHM décrit la facilité d'utilisation et l'efficacité d'une interface pour les utilisateurs. Il s'agit d'une mesure de la qualité de l'interface, qui peut être utilisée pour évaluer et améliorer les systèmes existants.

Les critères de l'utilisabilité sont :

- ⇒ Elle doit être facile à comprendre et à utiliser pour les utilisateurs
- ⇒ // // // efficace.
- ⇒ // // // satisfaisante.

3. Critères ergonomique :

- 3.1. La cohérence.
- 3.2. La concision.
- 3.3. Le retour d'informations.
- 3.4. La structuration des activités.
- 3.5. La flexibilité.
- 3.6. La gestion des erreurs.

Chapitre II : Conception des IHM

①. Pourquoi une méthode de conception IHM ?

- ✓ Réduction des risques et des coûts de maintenance.
- ✓ Réduction du budget / temps de formation.
- ✓ Attractivité de l'application, gain de productivité.
- ✓ Réutilisation et amélioration des composants de base.

②. Phases de Conception d'IHM : trois phases :

- ✓ Analyse : Facteurs humains.
- ✓ Développement : Prototypage Instrumentation.
- ✓ Évaluation : Etudes utilisateurs.

③. Méthode de conception pour l'IHM

3.1. Itérative : est une Méthodologie basée sur une succession de cycle composé des trois phases (analyse, Développement, Evaluation).

3.2. Incrementale : est une Méthodologie basée sur la réalisation d'une première partie, puis d'une seconde, etc...
• Travail sur une seule zone de l'interface.

• Travail sur l'intégralité de l'interface jusqu'à satisfaction.

• Cycle répété jusqu'à obtention d'une interface satisfaisante.

• Développement de solutions partielles, intermédiaires.

• Prise en compte de nouveaux objectifs.

• Prise en compte de l'avis des utilisateurs qui peuvent changer.

3.3. Prototypage : Obtenir une interface finale passe par plusieurs étapes :

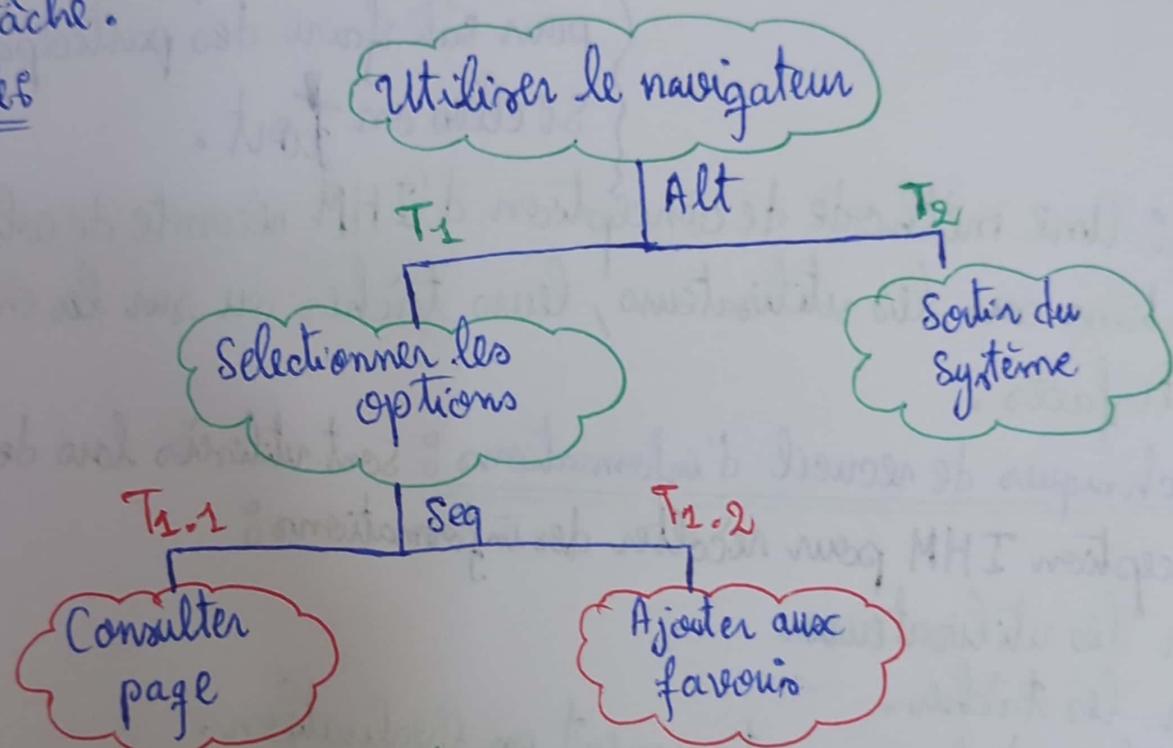
- ✓ Croquis (Sketch) : aperçu global de l'interface (itération générale).
- ✓ Maquette (mockup, wireframe) : interface détaillée (sans interaction).
- ✓ Prototype : version incomplète d'une interface (avec interactions).

3.4. Centrage utilisateur: il y'a trois modèles:

3.4.1. Modèle de l'utilisateur: identifier les caractéristiques pertinentes de l'utilisateur: Données générales (age, genre, niveau de formation, ...); Données liées à l'application (exemple: compétences sur le domaine, compétences en informatique et sur le système).

3.4.2. Modèle de la tâche: identifier l'enchaînement des processus d'une tâche.

Exemples



3.4.3. Modèle de l'interaction: établir une correspondance intuitive et "naturelle":

✓ Les objets conceptuels manipulés.

✓ La présentation et les interactions.

✓ Inspirée du monde réel.

3.5. Évaluation précoce: l'évaluation des croquis, maquettes, prototypes et interfaces est fréquente et intervient très tôt dans la conception.

3.6. Relations concepteur / utilisateur informative ou participative:

Utilisateur informatif

* L'utilisateur est intégré dans l'équipe de conception, mais ne participe pas aux choix finaux.

Utilisateur participatif

* L'utilisateur est intégré dans l'équipe de conception comme partenaire de conception à part entière, et participe donc aux choix de conception finaux.

* Obligation d'accepter des compromis pour satisfaire des participants, même si elles ont tort.

Enfin : Une méthode de conception d'IHM nécessite de collecter des informations sur les utilisateurs, leurs tâches ou sur les évaluations des interfaces.

1. Techniques de recueil d'informations sont utilisées lors de la méthode de conception IHM pour recueillir des informations

=> sur les utilisateurs.

=> Sur les tâches.

=> Sur les interfaces, notamment en évaluation.

5. Quelques notions

* Conception par prototypage consiste à concevoir des versions intermédiaires et incomplètes d'un logiciel ou d'un site web. Elle permet de tester l'utilisation d'un produit auprès d'utilisatrices.

* GOMS signifie "Goals, Operators, Methods and Selection rules" (Objectifs, Opérateurs, Méthodes et règles de sélection). Il est un modèle formel qui décrit comment l'utilisateurs interagissent avec un système informatique en décomposant le processus en une série d'objectif, d'opérateurs de méthode et de règles sélection.

GOMS utilisé pour analyser et évaluer les interfaces utilisatrices.

* Inspection cognitive est une méthode d'évaluation de l'utilisabilité d'une interface utilisateur en analysant les processus cognitifs utilisés par les utilisateurs pour interagir avec le système.