Conception et analyse du projet – Application ToDo Liste

Objectifs pédagogiques

À l'issue de cette séance, l'étudiant doit être capable de :

- Analyser des besoins exprimés en langage naturel.
- Identifier les fonctionnalités attendues d'une application.
- Concevoir des **diagrammes UML** pour représenter les cas d'usage et les classes.
- Organiser les composants du projet en couches logiques (interface, logique métier, données).
- Réaliser un prototype graphique simple à l'aide de **Tkinter**.

COURS

1. **Analyse fonctionnelle**

Avant d'écrire du code, il faut **analyser le besoin** :

"Je veux une application qui me permette de noter mes tâches à faire, de les afficher, de les modifier, et de les supprimer."

Il est fondamental de comprendre ce que l'application doit faire. C'est ce qu'on appelle **l'analyse des besoins fonctionnels**.

Exemple de cahier des charges minimal :

L'utilisateur souhaite une application dans laquelle il peut :

- Ajouter une tâche à faire
- Afficher toutes ses tâches
- Supprimer une tâche
- Marquer une tâche comme terminée
- (Optionnel : filtrer, trier, modifier une tâche)

On va s'appuyer sur cette description pour construire les **modèles UML**.

2. **Modélisation UML** – Cas d'utilisation (Use Case Diagram)

Le diagramme de **cas d'utilisation** permet de représenter les **fonctionnalités principales** du système, du point de vue de l'utilisateur.

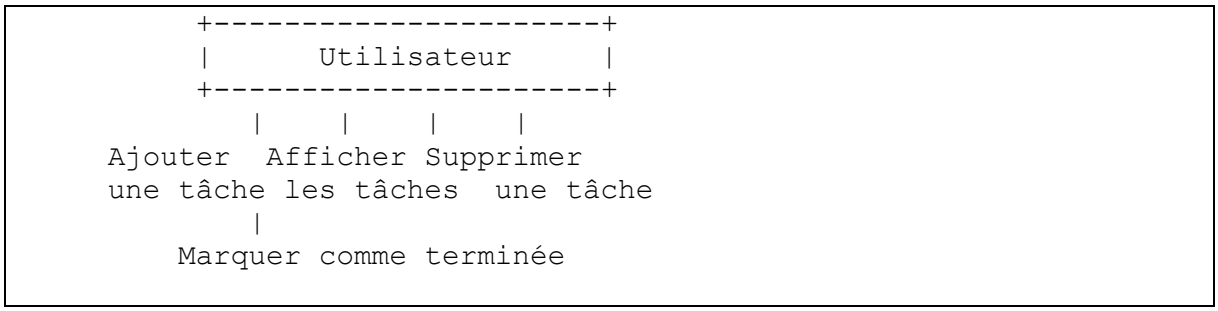
Acteur :

- `Utilisateur`

Cas d'utilisation :

- Ajouter une tâche
- Supprimer une tâche
- Afficher les tâches
- Marquer une tâche comme terminée

Exemple de diagramme version graphique simplifiée :



Version PlantUML (pour générer une image dans diagrams.net ou PlantUML) :

```
@startuml
actor Utilisateur

rectangle "Application ToDoList" {
  (Ajouter une tâche)
  (Supprimer une tâche)
  (Afficher les tâches)
  (Marquer comme terminée)
}

Utilisateur --> (Ajouter une tâche)
Utilisateur --> (Supprimer une tâche)
Utilisateur --> (Afficher les tâches)
Utilisateur --> (Marquer comme terminée)
@enduml
```

3. Modélisation UML – Diagramme de classes

Une fois les cas d'usage définis, on passe à la **modélisation du code** avec un **diagramme de classes** UML.

- **Classe `Tache`**

- **Attributs** :
  - `description : str`
  - `terminee : bool`
  - `date_creation : datetime`
- **Méthodes** :
  - `marquer_terminee()`
  - `__str__()` *(une **méthode spéciale (ou "magique")** qui définit la représentation sous forme de chaîne de caractères (string) d'un objet. Elle est appelée automatiquement lorsque vous utilisez la fonction str() ou print() sur un objet.)*

➢ **Classe `ToDoListe`**

- **Attributs** :
  - `taches : list`
- **Méthodes** :
  - `ajouter_tache(description)`
  - `supprimer_tache(index)`
  - `afficher_taches()`

Diagramme UML (texte) :

```
pgsql
+------------------+
|      Tache       |
+------------------+
| - description: str
| - terminee: bool
| - date_creation: datetime
+------------------+
| +marquer_terminee()
| +__str__()       |
+------------------+

+----------------------+
|      ToDoListe       |
+----------------------+
| - taches: list[Tache]|
+----------------------+
| +ajouter_tache()     |
| +supprimer_tache()   |
| +afficher_taches()   |
+----------------------+
```

Ces classes forment la **logique métier** de l'application.

4. Architecture logicielle (MVC)

MVC : Modèle – Vue – Contrôleur

- **Modèle** : Classes représentant les tâches (titre, date, état…).
- **Vue** : Interface utilisateur avec Tkinter (saisie, affichage, actions).

- **Contrôleur** : Coordination entre actions utilisateur et logique métier.

On structure le projet en **trois couches** :

Couche	Rôle
Interface utilisateur	**Saisie et affichage (Tkinter)**
Logique métier	**Gestion des objets Tache, via la programmation objet**
Données	**Stockage futur des tâches (ex. SQLite, JSON, etc.)**

TRAVAUX PRATIQUES

TP 1 – Document de conception UML

Objectif :

Créer une **modélisation UML complète** de l'application.

Travail demandé :

1. Rédiger une courte analyse fonctionnelle.
2. Créer un **diagramme de cas d'utilisation**.
3. Créer un **diagramme de classes UML**.

Livrable :

- Document PDF ou papier avec les deux diagrammes
- Utilisation possible d'outils comme https://app.diagrams.net ou PlantUML.

TP 2 – Prototypage de l'interface utilisateur

Objectif :

Créer une **interface simple** avec Tkinter pour :

- Ajouter une tâche
- L'afficher dans une liste

Code de base :

```
import tkinter as tk

def ajouter_tache():
    tache = champ_texte.get()
    if tache:
        liste.insert(tk.END, tache)
        champ_texte.delete(0, tk.END)

fenetre = tk.Tk()
fenetre.title("ToDo Liste")

champ_texte = tk.Entry(fenetre, width=40)
champ_texte.pack(pady=5)

btn_ajouter = tk.Button(fenetre, text="Ajouter",
command=ajouter_tache)
btn_ajouter.pack()

liste = tk.Listbox(fenetre, width=50)
liste.pack(pady=10)

fenetre.mainloop()
```

Ce prototype ne sauvegarde pas encore les données, mais permet de valider l'**ergonomie de base**.

Exercices supplémentaires

Exercice 1 : Ajouter une classe Tache avec une date

```
from datetime import datetime

class Tache:
    def __init__(self, description):
        self.description = description
        self.terminee = False
        self.date_creation = datetime.now()

    def marquer_terminee(self):
        self.terminee = True

    def __str__(self):
        statut = "✓" if self.terminee else "temps"
        return f"{statut} {self.description}
({self.date_creation.strftime('%d/%m/%Y')})"
```

Exercice 2 : Ajouter un bouton "Supprimer la tâche sélectionnée" à l'interface Tkinter

```
def supprimer_tache():
    selection = liste.curselection()
    if selection:
        liste.delete(selection[0])
```

```
btn_supprimer = tk.Button(fenetre, text="Supprimer",
command=supprimer_tache)
btn_supprimer.pack()
```