



# COUPLAGE FORT ET COUPLAGE FAIBLE

@med Mohammed Ezzaim

# INTRODUCTION

Spring Frameworks



## Objectif du cours:

Comprendre ce qu'est le couplage entre les classes en programmation orientée objet, et pourquoi le couplage faible est préférable pour créer des programmes flexibles, faciles à tester et à maintenir.

## Qu'est-ce que le Couplage ?

Le couplage représente le niveau de dépendance entre deux classes.

- Si une classe dépend fortement d'une autre : on parle de couplage fort.
- Si une classe est indépendante ou dépend peu des autres : c'est un couplage faible.

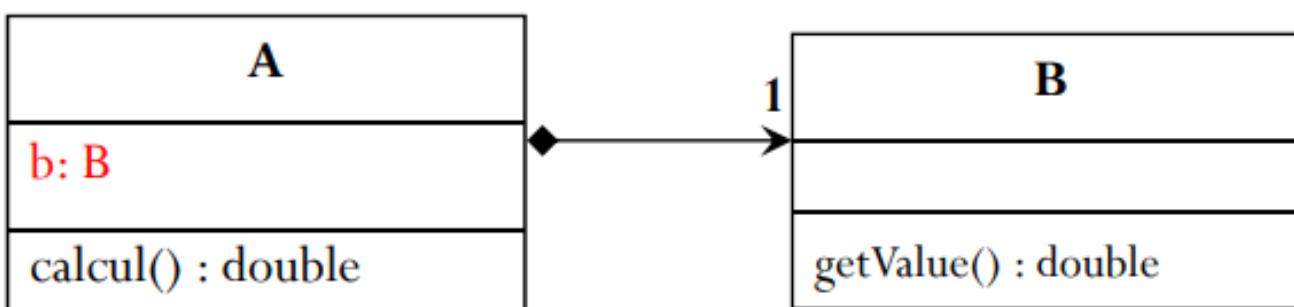
# COUPLAGE FORT

Spring Frameworks



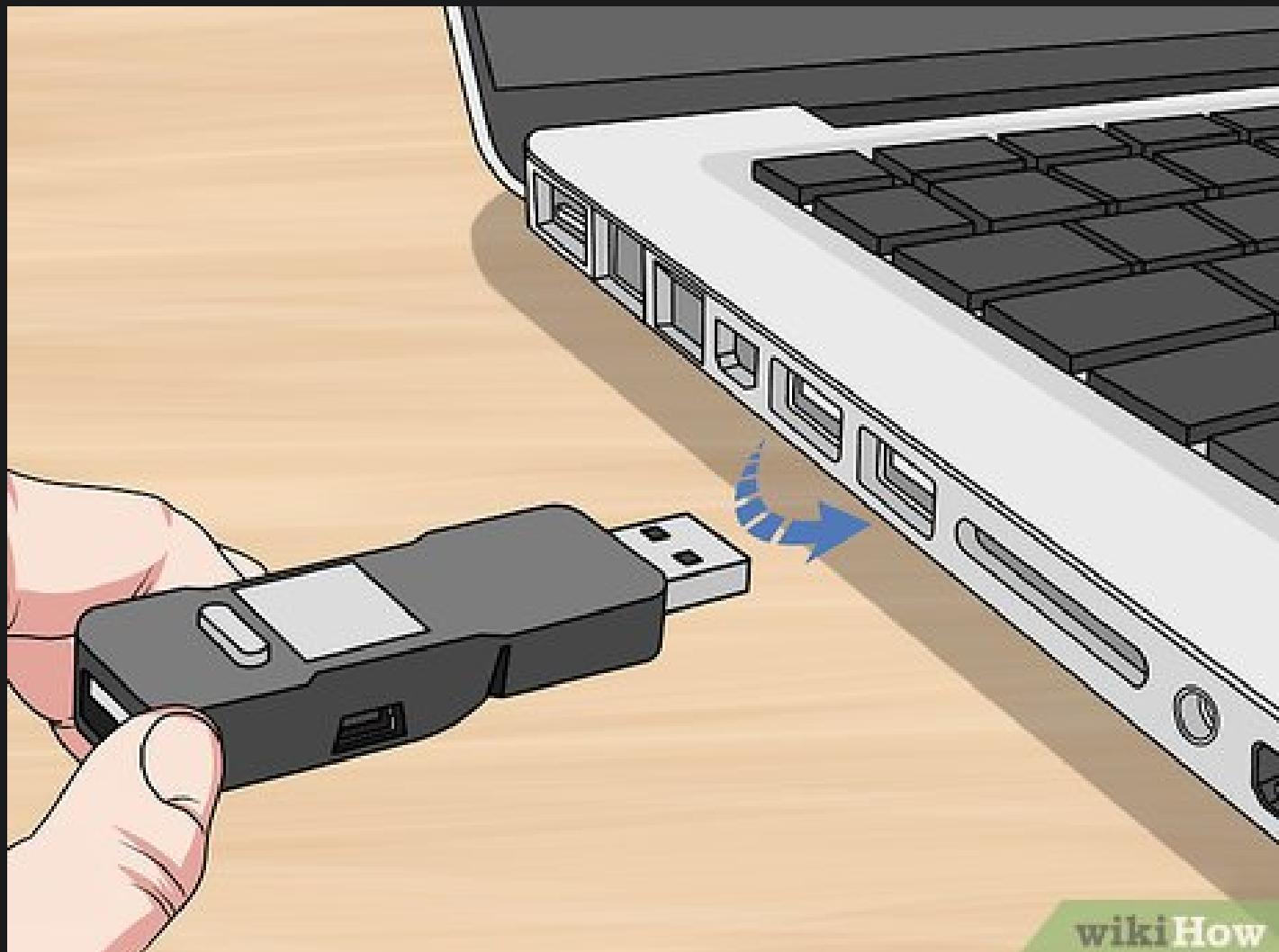
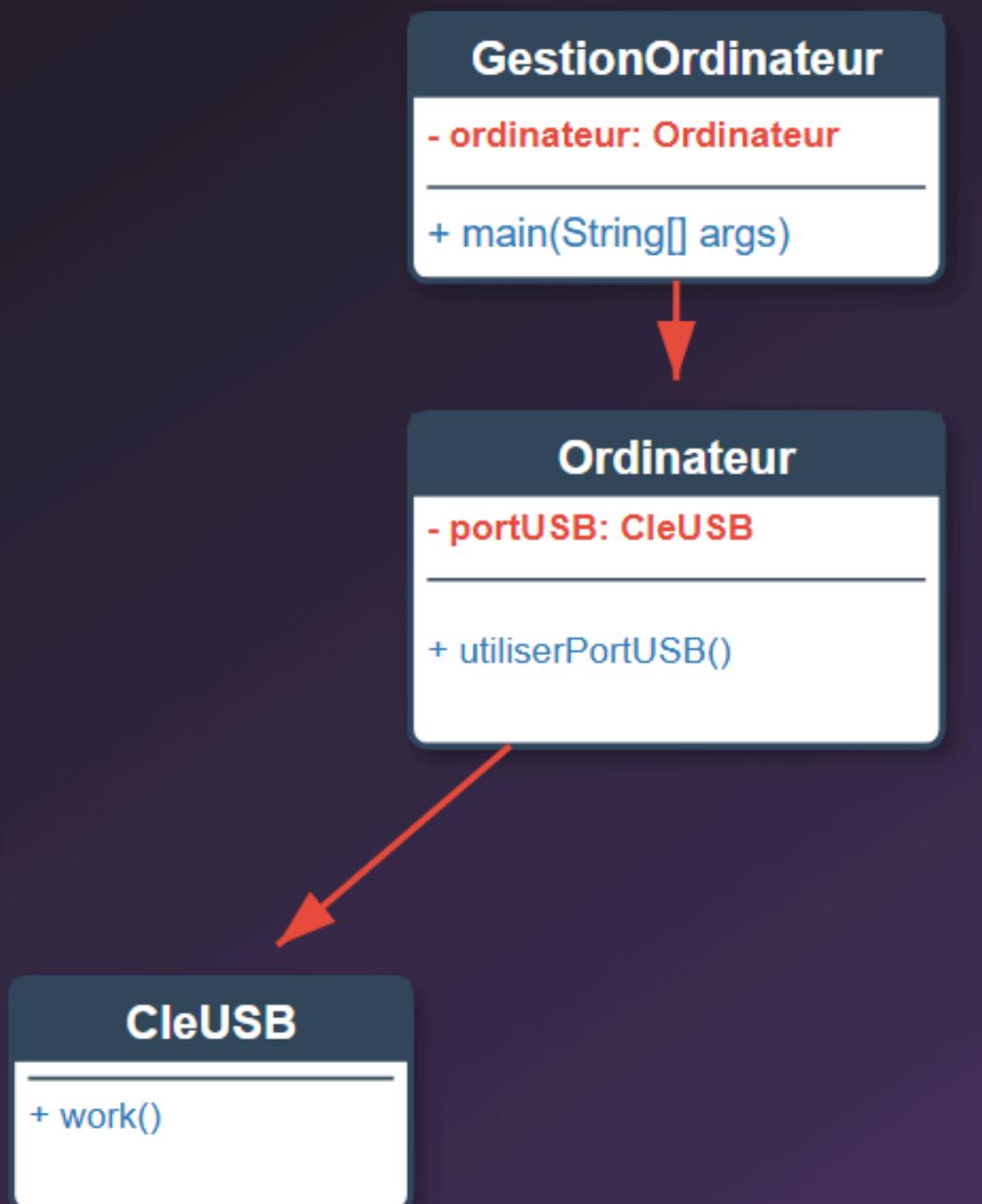
# Couplage fort

- Quand une classe A est lié à une classe B, on dit que la classe A est fortement couplée à la classe B.
- La classe A ne peut fonctionner qu'en présence de la classe B.
- Si une nouvelle version de la classe B (soit B2), est créée, on est obligé de modifier dans la classe A.
- Modifier une classe implique:
  - Il faut disposer du code source.
  - Il faut recompiler, déployer et distribuer la nouvelle application aux clients.
  - Ce qui engendre un cauchemar au niveau de la maintenance de l'application





## ✗ COUPLAGE FORT



wikiHow



```
● ● ●  
public class Ordinateur {  
    private final CleUSB portUSB;  
  
    public Ordinateur() {  
        this.portUSB = new CleUSB(); // Couplage fort: création directe de l'objet CleUSB  
    }  
  
    public void utiliserPortUSB(){  
        portUSB.work();  
    }  
}
```

```
● ● ●  
public class CleUSB {  
    public void work() {  
        System.out.println("La clé USB fonctionne");  
    }  
}
```

```
● ● ●  
public class GestionOrdinateur {  
    public static void main(String[] args) {  
        Ordinateur ordinateur = new Ordinateur();  
        ordinateur.utiliserPortUSB();  
    }  
}
```



## ✗ COUPLAGE FORT

GestionOrdinateur

- ordinateur: Ordinateur

+ main(String[] args)

Ordinateur

- portUSB: Clavier

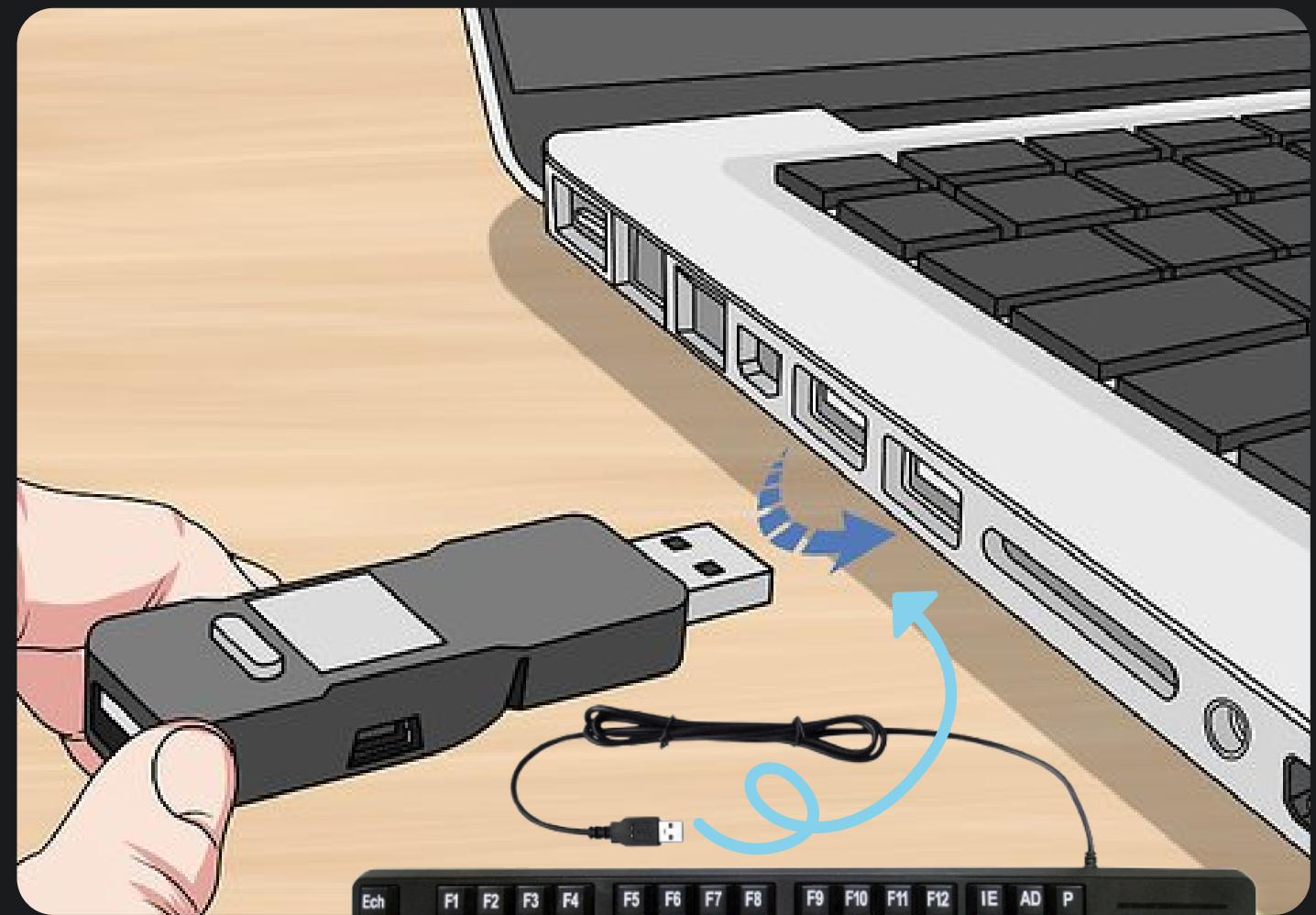
+ utiliserPortUSB()

CleUSB

+ work()

Clavier

+ work()





```
● ● ●

public class Ordinateur {
    private Clavier clavier; // Remplacement de CleUSB par Clavier

    public Ordinateur() {
        this.clavier = new Clavier(); // Couplage fort: création directe de Clavier
    }

    public void utiliserPortUSB() {
        clavier.work();
    }
}
```

```
● ● ●

public class Clavier {
    public void work() {
        System.out.println("Le clavier est en fonctionnement");
    }
}
```

```
● ● ●

public class GestionOrdinateur {
    public static void main(String[] args) {
        Ordinateur ordinateur = new Ordinateur();
        ordinateur.utiliserPortUSB();
    }
}
```



## Problèmes du couplage fort:

- Ce couplage fort n'a pas empêché de résoudre le problème au niveau fonctionnel.
- Mais cette conception nous ne a pas permis de créer une application fermée à la modification et ouverte à l'extension.
- L'Ordinateur **dépend directement de CleUSB** (concrète)
- → Impossible d'utiliser un autre périphérique (ex: Souris, DisqueDur) sans modifier le code source
- → La classe Ordinateur est rigidement couplée à CleUSB.
- → Impossible d'utiliser un autre périphérique sans modifier le code source.
- Modification du code existant :
  - Toutes les références à CleUSB doivent être remplacées par Clavier.
  - Risque d'introduire des erreurs si d'autres parties du système dépendaient de CleUSB.

# COUPLAGE FAIBLE

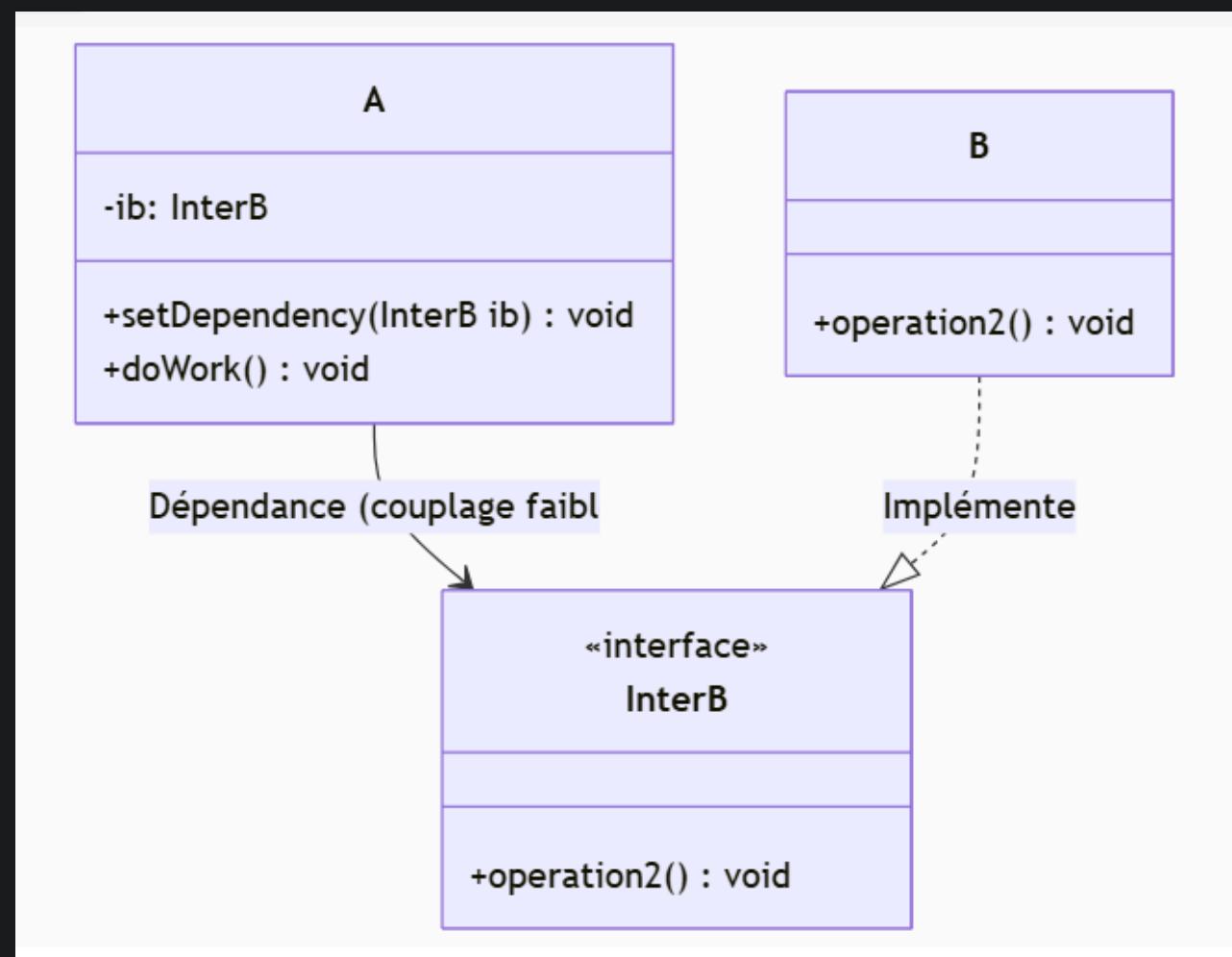
Spring Frameworks

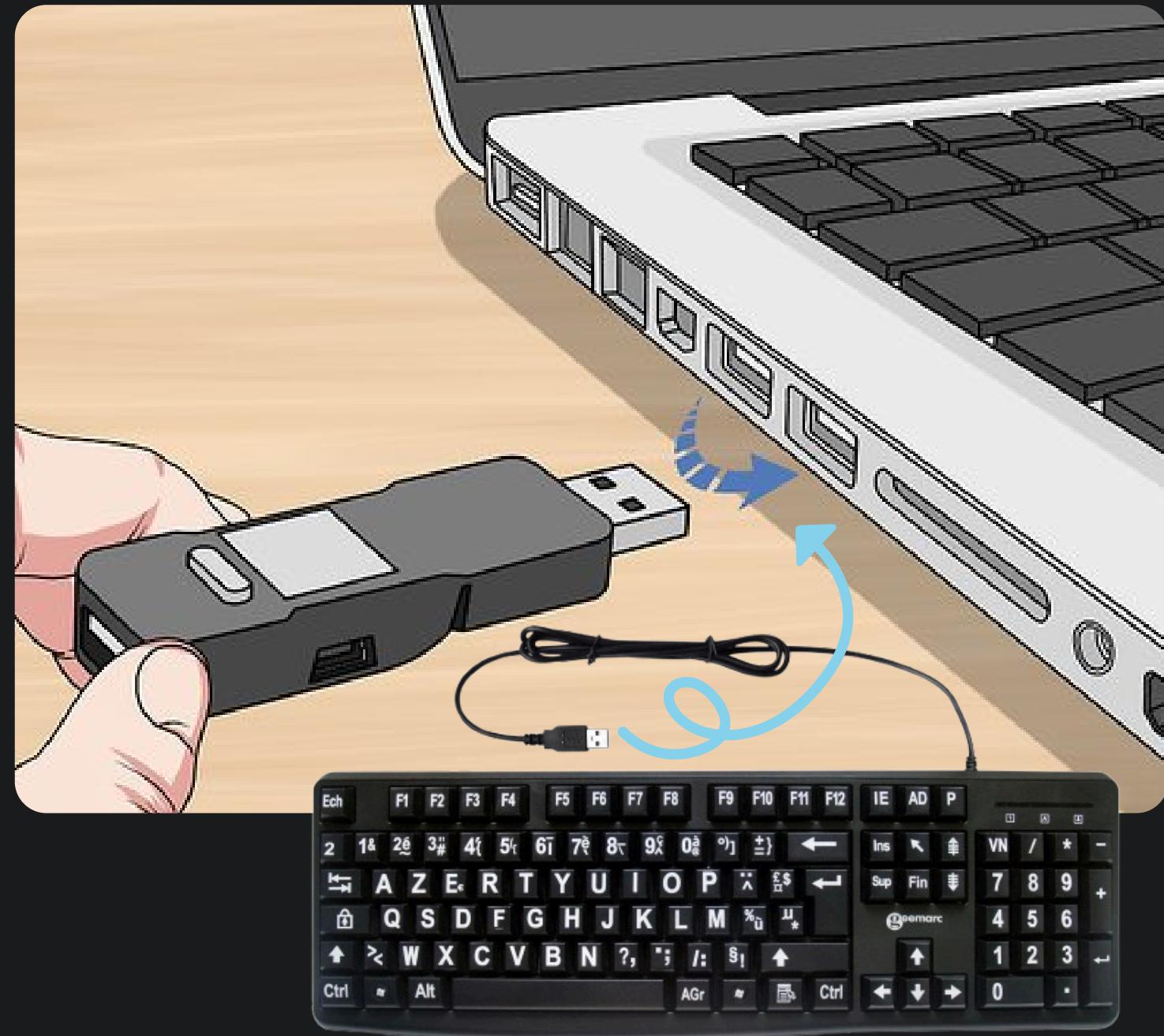
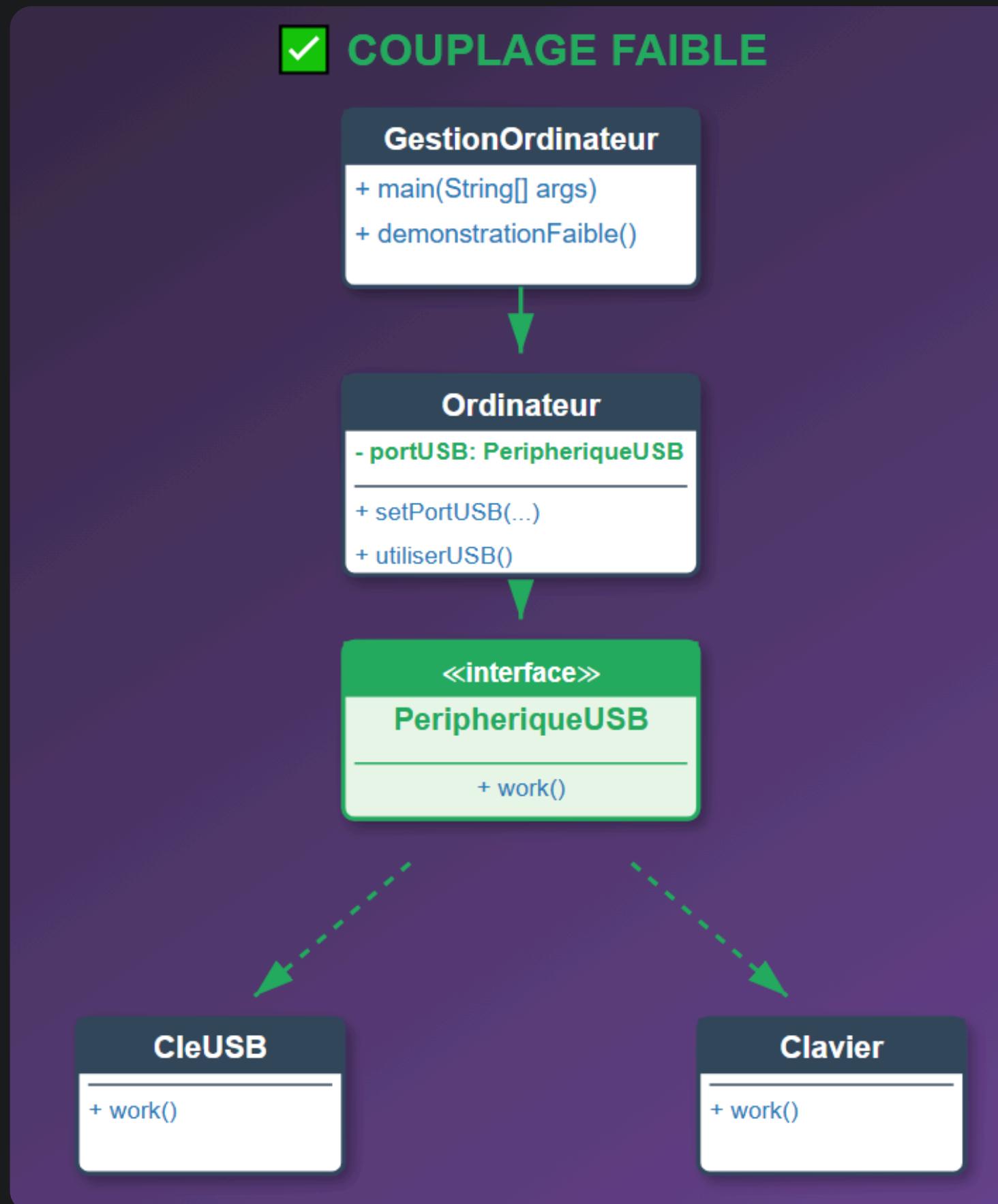


Ce diagramme illustre parfaitement le principe du couplage faible, un concept clé en conception logicielle qui favorise la modularité et la maintenabilité des systèmes. Voici les points essentiels à retenir pour votre rapport :

#### Définition du Couplage Faible :

Le couplage faible désigne une relation entre classes où une classe (A) dépend d'une abstraction (InterB) plutôt que d'une implémentation concrète (B). Cela permet une plus grande flexibilité en réduisant les dépendances directes entre composants.







```
interface PeripheriqueUSB {  
    void work();  
}
```



```
class CleUSB implements PeripheriqueUSB {  
    @Override  
    public void work() {  
        System.out.println("La clé USB stocke et lit des données");  
    }  
}
```



```
class Clavier implements PeripheriqueUSB {  
    @Override  
    public void work() {  
        System.out.println("Le clavier permet de saisir du texte");  
    }  
}
```



```
class Ordinateur {  
    private PeripheriqueUSB portUSB; // Dépendance abstraite  
  
    public void setPortUSB(PeripheriqueUSB peripherique) {  
        this.portUSB = peripherique;  
    }  
  
    public void utiliserUSB() {  
        if (portUSB != null) {  
            portUSB.work();  
        } else {  
            System.out.println("Aucun périphérique connecté");  
        }  
    }  
}
```



```
public class GestionOrdinateur {  
    public static void main(String[] args) {  
        demonstrationFaible();  
    }  
  
    public static void demonstrationFaible() {  
        // Création des périphériques  
        PeripheriqueUSB cle = new CleUSB();  
        PeripheriqueUSB clavier = new Clavier();  
  
        // Configuration de l'ordinateur  
        Ordinateur ordi = new Ordinateur();  
  
        // Branchement alternatif des périphériques  
        System.out.println("== Utilisation clé USB ==");  
        ordi.setPortUSB(cle);  
        ordi.utiliserUSB();  
  
        System.out.println("\n== Utilisation clavier ==");  
        ordi.setPortUSB(clavier);  
        ordi.utiliserUSB();  
  
        // On pourrait facilement ajouter d'autres périphériques  
        // sans modifier les classes existantes  
    }  
}
```



- Ce code illustre parfaitement les avantages du couplage faible en programmation orientée objet.
- Contrairement au couplage fort, où les classes dépendent directement d'implémentations concrètes, cette approche repose sur une abstraction (`PeripheriqueUSB`), permettant une meilleure modularité et flexibilité.
- L'ordinateur n'est plus lié à des périphériques spécifiques comme `CleUSB` ou `Clavier`, mais fonctionne avec n'importe quel objet implémentant l'interface. Cela facilite l'extension du système sans modifier le code existant, respectant ainsi le principe ouvert/fermé (OCP).

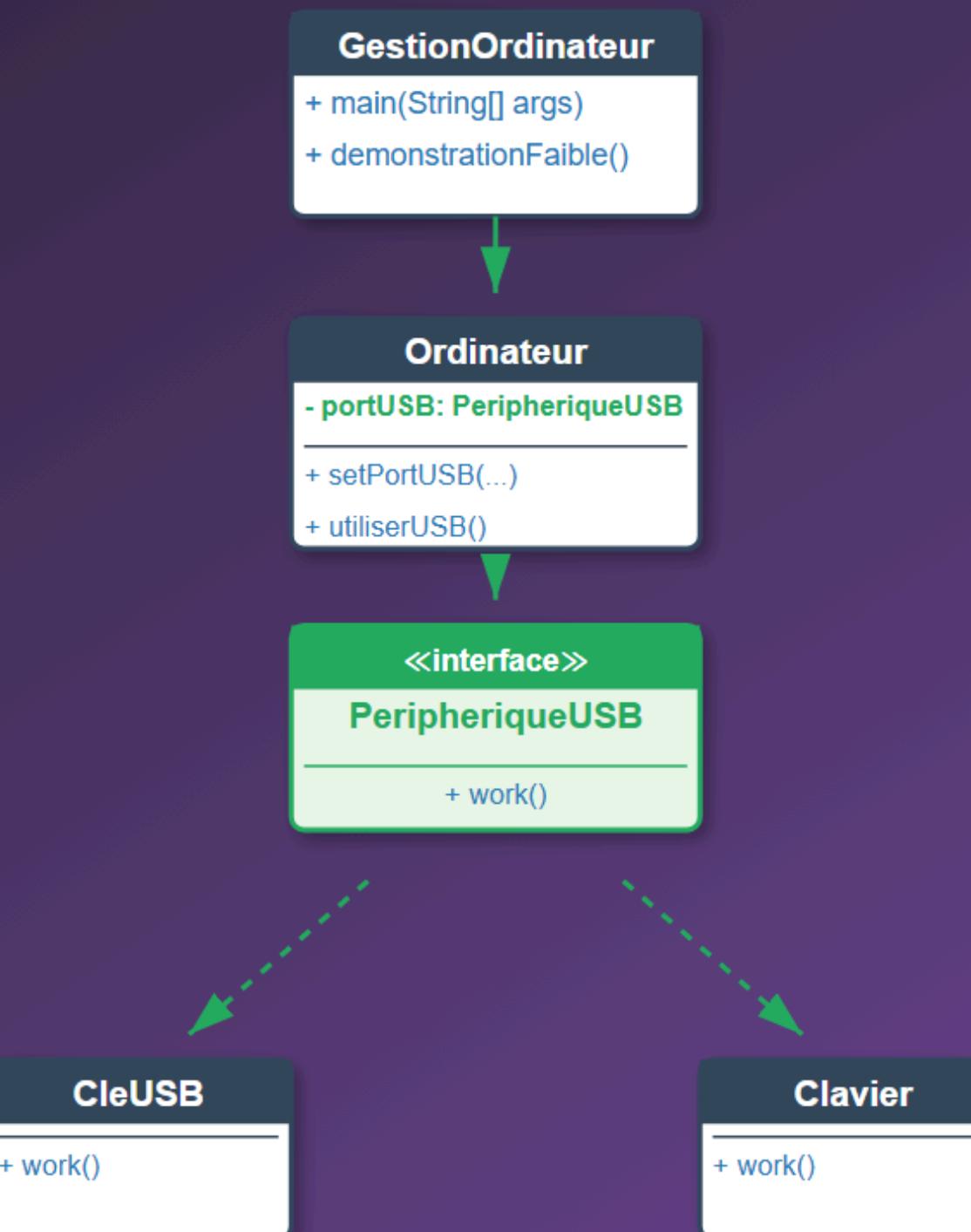


## Diagrammes UML - Couplage Fort vs Couplage Faible

### ✗ COUPLAGE FORT



### ✓ COUPLAGE FAIBLE





```
src
  app
    GestionOrdinateur
    GestionOrdinateurV2
  bean
    Clavier
    CleUSB
    PeripheriqueUSB
    Souris
  ord
    Ordinateur
```

```
/*
 * @author Ezzaim Mohammed
 */
public interface PeripheriqueUSB {
    void work();
}
```

```
public class CleUSB implements PeripheriqueUSB{
    @Override
    public void work() {
        System.out.println("Le Cle USB fonctionne");
    }
}
```

```
public class Clavier implements PeripheriqueUSB{
    @Override
    public void work() {
        System.out.println("La clavier fonctionne");
    }
}
```

```
public class Souris implements PeripheriqueUSB{
    @Override
    public void work() {
        System.out.println("Souris fonctionne");
    }
}
```



```
● ● ●

/**
 * @author Ezzaim Mohammed
 */

public class Ordinateur {
    private PeripheriqueUSB portUSB;

    public Ordinateur() {
    }

    public void utiliserPortUSB(){
        portUSB.work();
    }

    public void setPeripheriqueUSB(PeripheriqueUSB portUSB){
        this.portUSB = portUSB;
    }

}
```



```
/**  
 * @author Ezzaim Mohammed  
 */  
public class GestionOrdinateurV2 {  
    public static void main(String[] args) throws FileNotFoundException, ClassNotFoundException, InstantiationException,  
IllegalAccessException {  
    Scanner scanner = new Scanner(new File("config.txt"));  
  
    String peripheriqueClassName = scanner.nextLine();  
    Class peripheriqueClass = Class.forName(peripheriqueClassName);  
    PeripheriqueUSB peripheriqueObject = (PeripheriqueUSB) peripheriqueClass.newInstance();  
  
    String ordinateurClassName = scanner.nextLine();  
    Class ordinateurClass = Class.forName(ordinateurClassName);  
    Ordinateur ordinateur = (Ordinateur) ordinateurClass.newInstance();  
    ordinateur.setPeripheriqueUSB(peripheriqueObject);  
  
    ordinateur.utiliserPortUSB();  
}  
}
```

# MERCI !

@med Mohammed Ezzaim