Couplage fort/faible

Démonstration 1 du module 3

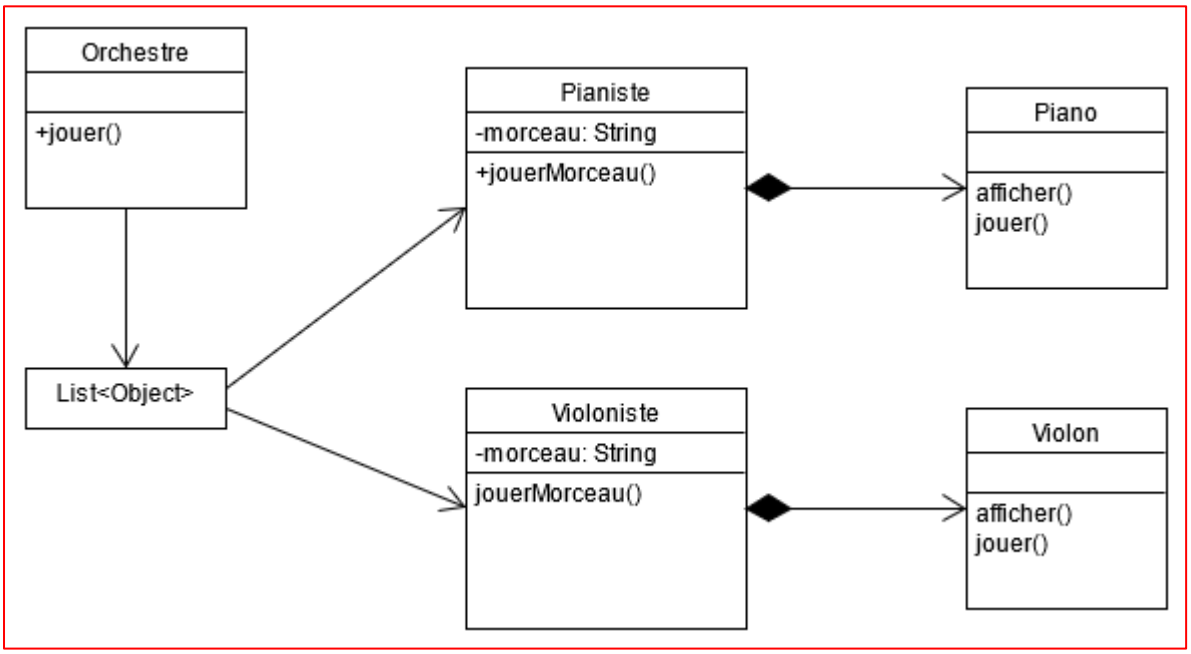
L'objectif de cette démonstration est de mettre en avant les avantages du couplage faible.

Déroulement

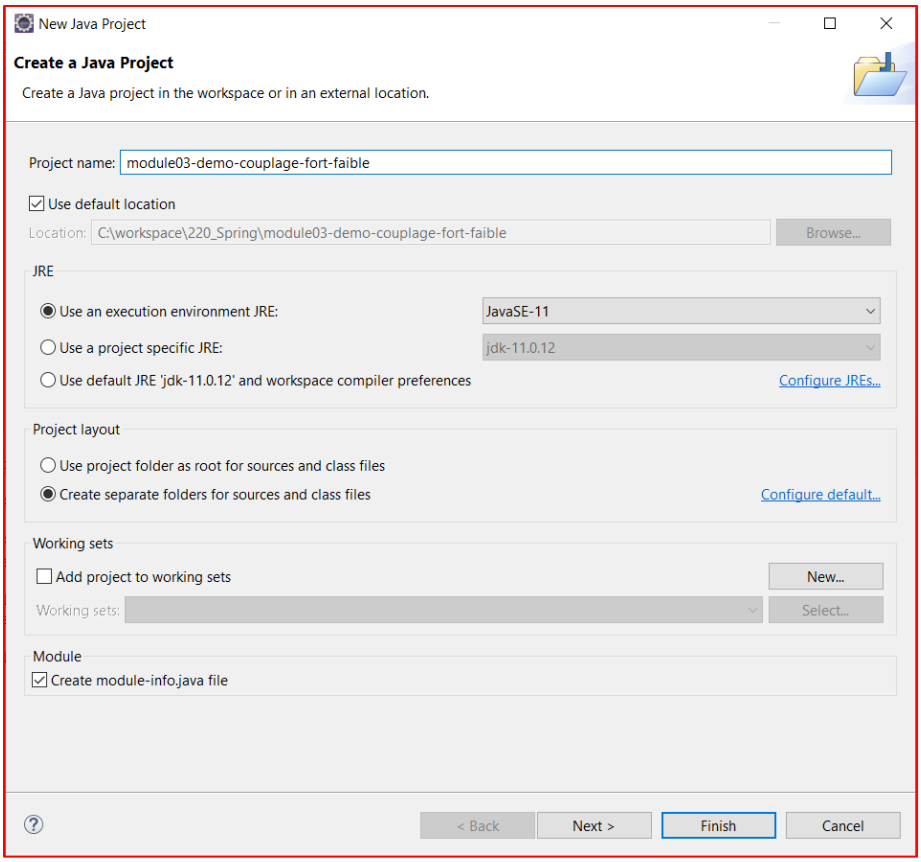
Couplage fort

Pour illustrer la notion de couplage fort, prenons l'exemple d'un orchestre.

- Un orchestre fera référence à une liste de musiciens.
- Nous définissons 2 types de musiciens : les pianistes et les violonistes.



- Créer un projet Java 11, représentant ce diagramme :



- La classe Piano :

```
package fr.eni.demo.orchestre;

public class Piano {
	public void afficher() {
		System.out.println("Je suis un piano...");
	}
	public void jouer() {
		System.out.println("LA LA LA");
	}
}
```

- La classe Pianiste propose le comportement jouerMorceau() et est composée d'un piano.

```
package fr.eni.demo.orchestre;

public class Pianiste {

	private Piano piano;
	private String morceau;

	public Pianiste( String morceau) {
		this.piano = new Piano();
		this.morceau = morceau;
	}

	public void jouerMorceau() {
		piano.afficher();
		System.out.println("et je joue le morceau " + morceau);
		piano.jouer();
	}
}
```

- La classe Piano :

```
package fr.eni.demo.orchestre;

public class Violon {
	public void afficher() {
		System.out.println("Je suis un violon...");
	}
	public void jouer() {
		System.out.println("ZIN ZIN ZIN");
	}
}
```

- La classe Violoniste propose le même comportement jouerMorceau() que la classe Pianiste est composée d'un violon.

```
package fr.eni.demo.orchestre;

public class Violoniste {
	private Violon violon;
	private String morceau;

	public Violoniste( String morceau) {
		super();
		this.violon = new Violon();
		this.morceau = morceau;
	}

	public void jouerMorceau() {
		violon.afficher();
		System.out.println("et je joue le morceau " + morceau);
		violon.jouer();
	}
}
```

- La classe orchestre est composée d'une liste d'objets qui sont soit des pianistes soit des violonistes :
  - Et une méthode jouer qui permet d'appeler le comportement de jouerMorceau sur Pianiste ou Violoniste

```
public class Orchestre {
	private List<Object> listeMusiciens;

	public Orchestre() {
		listeMusiciens = new ArrayList<>();
	}

	public void ajout(Object musicien) {
		listeMusiciens.add(musicien);
	}

	public void jouer() {
		for(Object object: listeMusiciens) {
			if(object instanceof Pianiste) {
				((Pianiste) object).jouerMorceau();
			}
			if(object instanceof Violoniste) {
				((Violoniste) object).jouerMorceau();
			}
		}
	}
}
```

- La classe d'exécution de l'application :

```
package fr.eni.demo.orchestre;

public class DemoApplication {
	public static void main(String[] args) {
		System.out.println("Le pianiste : ");
		Pianiste pianiste = new Pianiste("La 9eme de Beethoven");
		pianiste.jouerMorceau();

		System.out.println("Le violoniste : ");
		Violoniste violoniste = new Violoniste("La 9eme de Beethoven");
		violoniste.jouerMorceau();

		System.out.println("L'orchestre : ");
		Orchestre orchestre = new Orchestre();
		orchestre.ajout(pianiste);
		orchestre.ajout(violoniste);
		orchestre.jouer();
	}
}
```

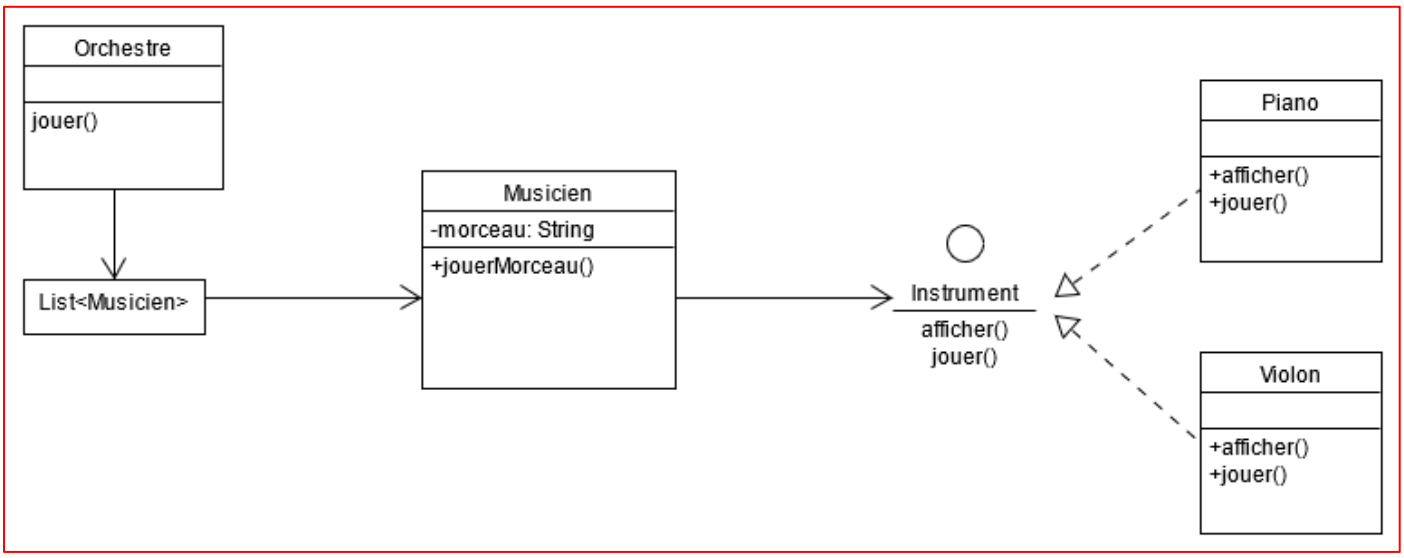
- Il y a plusieurs couplages forts dans ce diagramme :
  - La classe Pianiste dépend fortement de son association avec Piano.
  - Si il y avait une modification de la classe Piano, par exemple la modification du nom de la méthode jouer, la méthode jouerMorceau() de Pianiste ne fonctionnerait plus.
  - On dit que les 2 classes sont liées fortement.

- Le problème est identique avec le Violoniste fortement dépendant de son Violon. Un changement de la classe Violon ou sa suppression auront un impact sur la classe Violoniste. De plus un Violoniste ne peut ici jouer que de l'implémentation de Violon proposée, alors qu'il existe une grande variété de violons différents.

- Enfin, la classe orchestre utilise aussi un couplage fort pour définir les membres de l'orchestre.
  - En effet, cette classe fait appel directement aux classes Pianiste et Violoniste. Le couplage fort empêche d'ajouter d'autres types de musiciens facilement, il faudra modifier la classe Orchestre pour le faire.

Couplage faible

- Modifier l'application actuelle en considérant les remarques précédentes.



- Ajout d'une interface Instrument, elle permet de référencer les comportements possibles sur les instruments de musique

```
package fr.eni.demo.orchestre;

public interface Instrument {
	void afficher();

	void jouer();
}
```

- Les 2 classes Piano et Violon doivent implémenter cette interface.
  - Elles sont maintenant cachées par cette interface, et si l'application évolue et qu'il faut ajouter la classe Trompette, l'architecture vers Musicien ne change pas il y aura seulement un nouveau type d'Instrument

```
package fr.eni.demo.orchestre;

public class Piano implements Instrument {
	public void afficher() {
		System.out.println("Je suis un piano...");
	}

	public void jouer() {
		System.out.println("LA LA LA");
	}
}
```

```
package fr.eni.demo.orchestre;

public class Violon implements Instrument {
	public void afficher() {
		System.out.println("Je suis un violon...");
	}
	public void jouer() {
		System.out.println("ZIN ZIN ZIN");
	}
}
```

- Création d'une classe Musicien faisant référence à l'interface : couplage faible.
    - Possible de créer tous types d'instruments et de les manipuler en tant que musicien.

```
package fr.eni.demo.orchestre;

public class Musicien {
	private Instrument instrument;
	private String morceau;

	public Musicien(String morceau, Instrument instrument) {
		this.instrument = instrument;
		this.morceau = morceau;
	}

	public void jouerMorceau() {
		instrument.afficher();
		System.out.println("et je joue le morceau " + morceau);
		instrument.jouer();
	}
}
```

- Modification de la classe Orchestre, elle est composée d'une liste de Musicien à présent.
    - Simplification de la méthode jouer. Il n'est plus nécessaire de transtyper un objet en un certain musicien (Pianiste, Violoniste). Il suffit d'appeler la méthode jouerMorceau de la classe Musicien.

```
package fr.eni.demo.orchestre;

public class Orchestre {
	private List<Musicien> listeMusiciens;

	public Orchestre() {
		listeMusiciens = new ArrayList<>();
	}

	public void ajout(Musicien musicien) {
		listeMusiciens.add(musicien);
	}

	public void jouer() {
		for (Musicien musicien : listeMusiciens) {
			musicien.jouerMorceau();
		}
	}
}
```

- Modification de la classe d'exécution de l'application :

```
package fr.eni.demo.orchestre;

public class DemoApplication {
	public static void main(String[] args) {
		System.out.println("Le pianiste : ");
		Musicien pianiste = new Musicien("La 9eme de Beethoven", new Piano());
		pianiste.jouerMorceau();

		System.out.println("Le violoniste : ");
		Musicien violoniste = new Musicien("La 9eme de Beethoven", new Violon());
		violoniste.jouerMorceau();

		System.out.println("L'orchestre : ");
```

```
                Orchestre orchestre = new Orchestre();
                orchestre.ajout(pianiste);
                orchestre.ajout(violoniste);
                orchestre.jouer();
        }
}
```

Avantages de la version faiblement couplée :

- Les instances de Musicien utilisent un instrument via le type Instrument et non plus le type d'implémentation.
- Cela donne la souplesse de créer de nouveaux instruments ou de modifier des instruments existants sans aucun impact sur la classe Musicien.