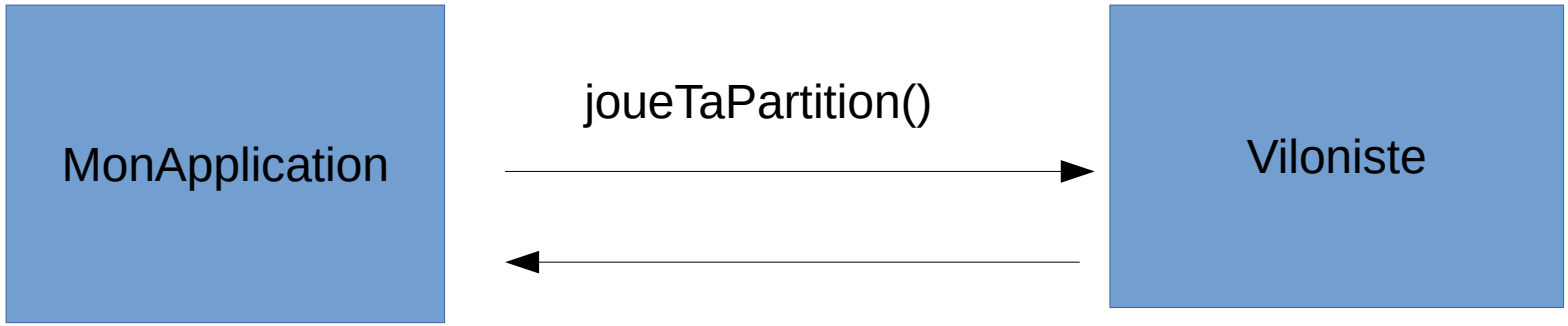
Cours Spring

02 – 01 Bonne pratiques de développement

Spécification foncionnelles

- on va coder une application Java qui va permettre de réaliser le scénario suivant :



- L'application doit être configurable , c'est à dire que l'application doit pouvoir fonctionner si on change de musicien (Pianniste, flutiste ...etc) par la suite (évolution future facilitée).

Les composants clés

- Une première application basique sans configuration:
  - une classe `MonApplication` avec un main()
  - Une classe `Violoniste` représentant un violoniste

- Prise en compte des "requirements" (aka besoin exprimé) :
  - une classe `MonApplication` avec un main()
  - Une classe `Violoniste` représentant un violoniste
  - Une classe `Pianiste` représentant un pianiste
  - Une interface `Musicien` représentant un musicien

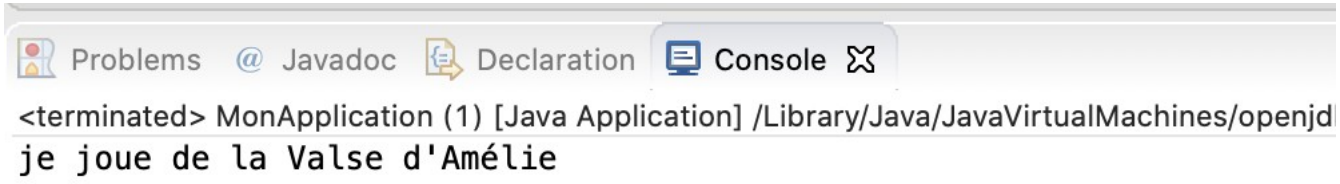
Un code fermé à la modification

```java
package com.springdemo;

public class MonApplication {

	public static void main(String[] args) {
		// create objet
		Violoniste musicien = new Violoniste();

		//use the objet
		System.out.println( musicien.jourTaPartition());
	}
}
```

```java
package com.springdemo;

public class Violoniste {
 public String joueTaPartition(){
		return "je joue de la Valse d'Amélie";
		}
 }
```

Pour info : On parle de couplage fort entre les objets.

A l'exécution on obtiens =>



Un code évolutif

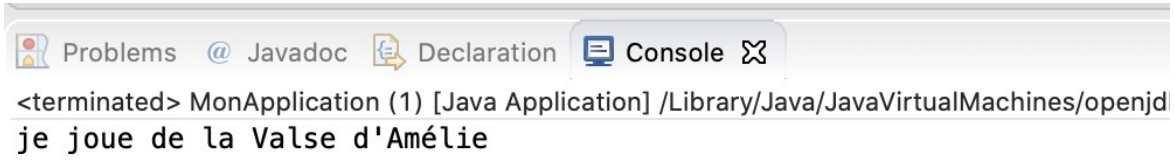
```java
package com.springdemo;

public class MonApplication {

	public static void main(String[] args) {
		// create objet
		Musicien musicien = new Violoniste();

		//use the objet
		System.out.println( musicien.joueTaPartition());
	}
}
```

```java
package com.springdemo;

public class Violoniste implements Musicien {
	@Override
	public String joueTaPartition(){
		return "je joue de la Valse d'Amélie";
	}
 }
```

```java
package com.springdemo;

public interface Musicien {

	public String joueTaPartition();
}
```

Ici on a un couplage faible

Si on exécute le résultat est le même =>



mais ce code fonctionnera pour d'autres musiciens.

Un code qui évolue par extension

```java
package com.springdemo;

public class MonApplication {

	public static void main(String[] args) {
		// create objet
		Musicien musicien = new Violoniste();

		//use the objet
		System.out.println( musicien.joueTaPartition());
	}
}
```

```java
package com.springdemo;

public interface Musicien {

	public String joueTaPartition();
}
```

```java
package com.springdemo;

public class Pianiste implements Musicien{

	@Override
	public String joueTaPartition() {

		return "je joue du piano debout";
	}

}
```

```java
package com.springdemo;

public class Violoniste implements Musicien {
	@Override
	public String joueTaPartition(){
		return "je joue de la Valse d'Amélie";
	}
 }
```

Point de départ de ce cours.

configurable != hard-coded

La mise à jour est facilitée,
mais elle n' est toujours pas configurable.