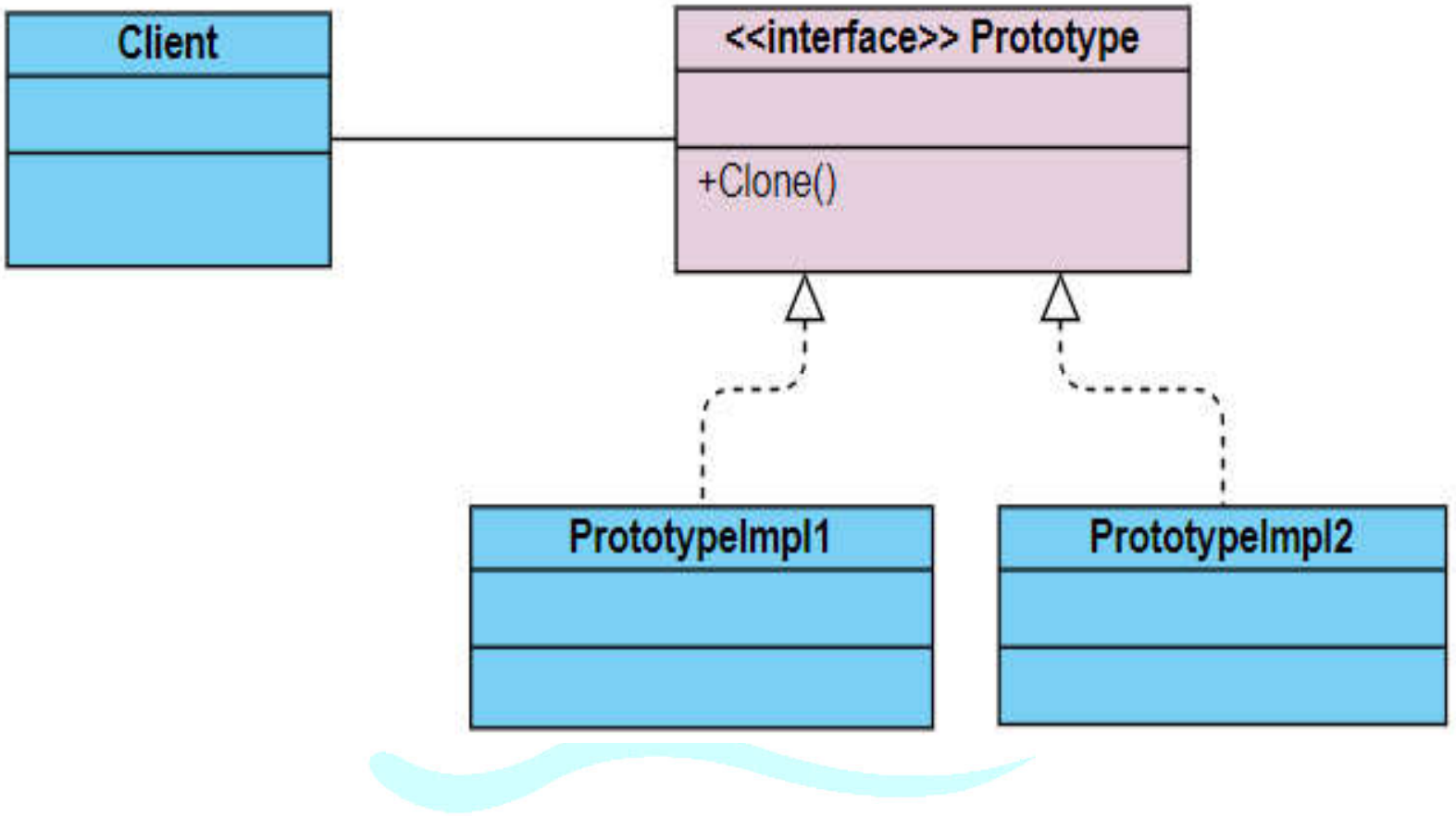
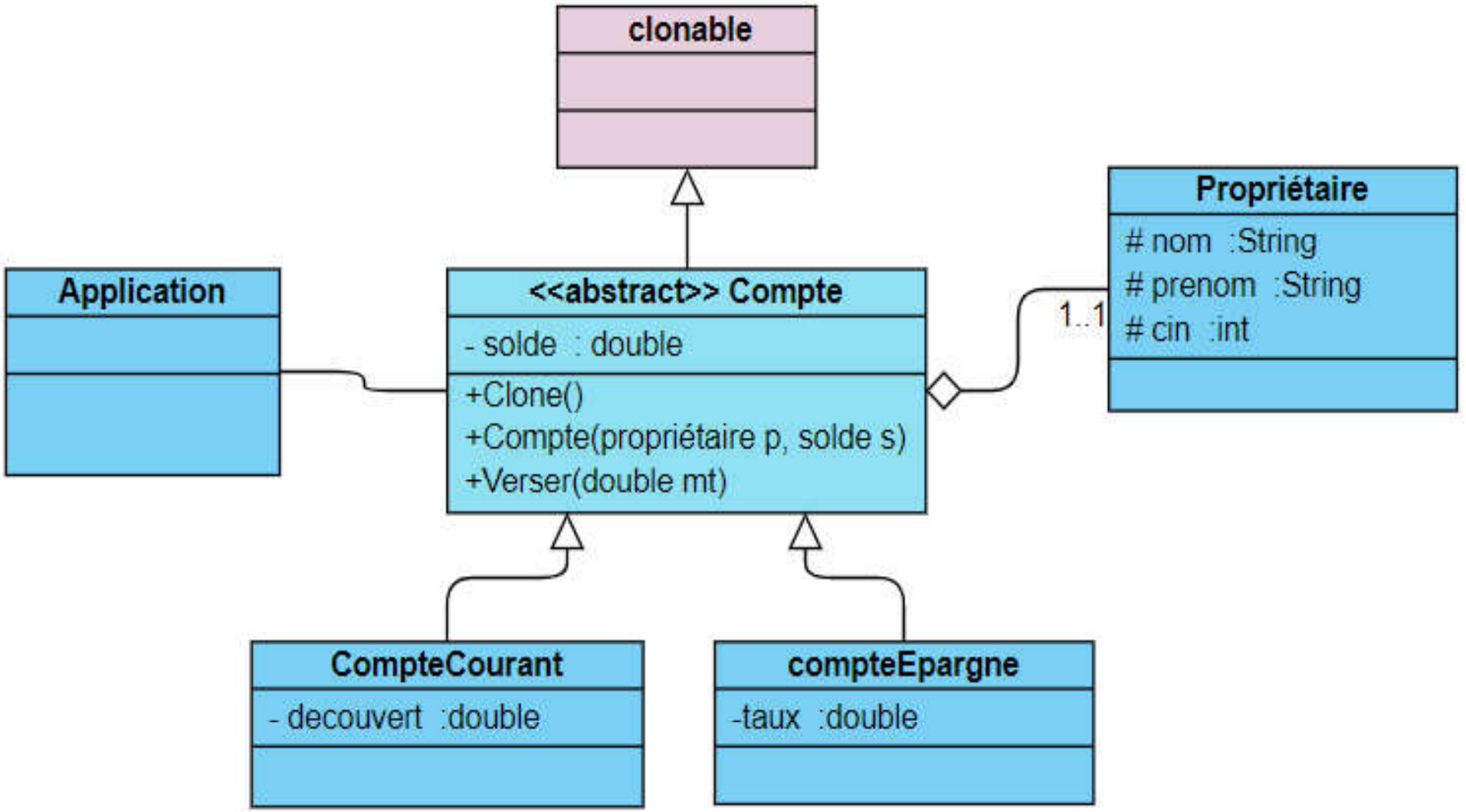
Pattern Prototype

- Catégorie : Création
- Objectif:
  - Fournir de nouveaux objets par la copie d'un exemple plutôt que de produire de nouvelles instance non initialisées d'une classes
- Raisons d'utilisation :
  - Cloner des objets sans les coupler à leurs classes concrètes et donc produire des objets complexes plus facilement. En effet, il est envisageable de créer de nouveaux objets à l'aide d'appels de constructeur. Cependant, lorsque vous voulez développer des applications d'entreprise, il peut s'avérer difficile de créer des nouveaux objets pour chaque client (problème de communication réseau, de lecture de base de données, etc…). Créer un prototype et le copier permet d'effectuer ces étapes une seule fois.

Pattern Prototype : Diagramme de classe



Pattern Prototype : Implémentation



Implémentation Java du pattern Prototype

```
public abstract class Compte implements Cloneable{
    protected propriétaire p;
    protected double  solde;
    public Compte(propriétaire p, double solde) {
        super();
        this.p = p;
        this.solde =solde;
    }
    public void verser(double mt){
        solde = solde+mt;
    }
    @Override
    protected Compte clone() throws CloneNotSupportedException {
        Compte c = (Compte) super.clone();
        return c;
    }
    @Override
    public String toString() {
        return "Compte [p=" + p + ", solde=" + solde + "]";
    }
    public void setSolde(double solde) {
        this.solde = solde;
    }
}
```

Implémentation Java du pattern Prototype

```
public class CompteCourant extends Compte{
    protected double decouvert;
    public CompteCourant(propriétaire p, double solde, double decouvert) {
        super(p, solde);
        this.decouvert = decouvert;
    }
    @Override
    public String toString() {
        return "CompteCourant [decouvert=" + decouvert + ", p=" + p
                + ", solde=" + solde + "]";
    }
}
```

```
public class CompteEpargne extends Compte{
    protected int taux;
    public CompteEpargne(propriétaire p, double solde, int taux) {
        super(p, solde);
        this.taux = taux;
    }
    @Override
    public String toString() {
     return "CompteEpargne[taux=" + taux + ", p=" + p + ", solde=" + solde
                + "]";
    }
}
```

Implémentation Java du pattern Prototype

```java
public class propriétaire{
    protected String Nom;
    protected String prenom;
    protected int cin;
    public propriétaire() {
        // TODO Auto-generated constructor stub
    }
    public propriétaire(String nom, String prenom, int cin) {
        super();
        Nom = nom;
        this.prenom = prenom;
        this.cin = cin;
    }
    @Override
    public String toString() {
        return "propriétaire [Nom=" + Nom + ", prenom=" + prenom + ", cin="
                + cin + "]";
    }
}
```

Implémentation Java du pattern Prototype

```
public class Application {

    public static void main(String[] args) throws CloneNotSupportedException {
      Compte c1 = new CompteCourant(new propriétaire("elhajjamy", "oussama", 123), 100000, 2000);
      Compte c2 = new CompteCourant(new propriétaire("Alaoui", "Amine", 124), 3000, 500);

      System.out.println(c1);
      System.out.println(c2);

      System.out.println("copie de c1");
      Compte c3 = c1.clone();
      System.out.println(c3);
      System.out.println("copie de c2");
      Compte c4 = c2.clone();
      System.out.println(c4);
      c1.p.Nom = "ali";
      System.out.println(c1);
      System.out.println(c3);
    }
}
```

Design patterns de GOF (Gang Of Four) (Gamma, Helm, Johnson, vlissides)

		Catégorie		
		Création	**Structure**	**Comportement**
Portée	**Classe**	Factory Method	Adapter	Interpreter
				Template Method
	Objet	Abstract Factory	Adapter	Chain of Responsibility
		Builder	Bridge	Command
		Prototype	Composite	Iterator
		Singleton	Decorator	Mediator
			Facade	Memento
			Flyweight	Observer
			Proxy	State
				Strategy
				Visitor