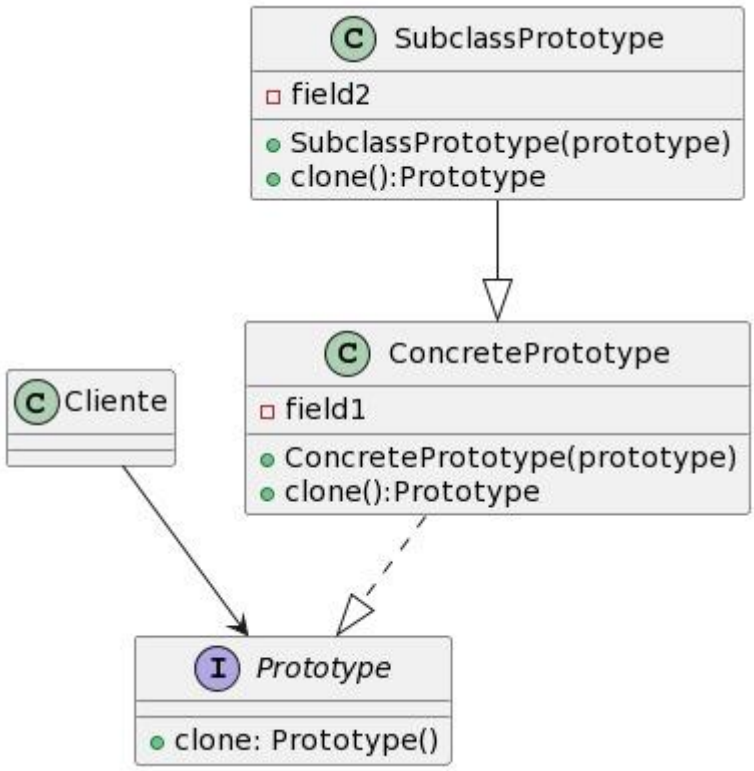
Prototype

```
public class Pessoa {

private Long altura;
private Long peso;
private String nome;

public Pessoa() {

}

public Long getAltura() {
return altura;
}

public void setAltura(Long altura) {
this.altura = altura;
}

public Long getPeso() {
return peso;
}

public void setPeso(Long peso) {
this.peso = peso;
}

public String getNome() {
return nome;
}
```

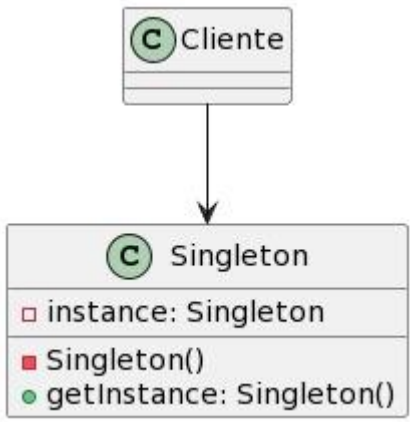
```
public void setNome(String nome) {
this.nome = nome;
}

public Pessoa clonar() {
Pessoa pClone = new Pessoa();
pClone.setAltura(this.altura);
pClone.setPeso(this.peso);
pClone.setNome(this.nome);
return pClone;
}

}
```

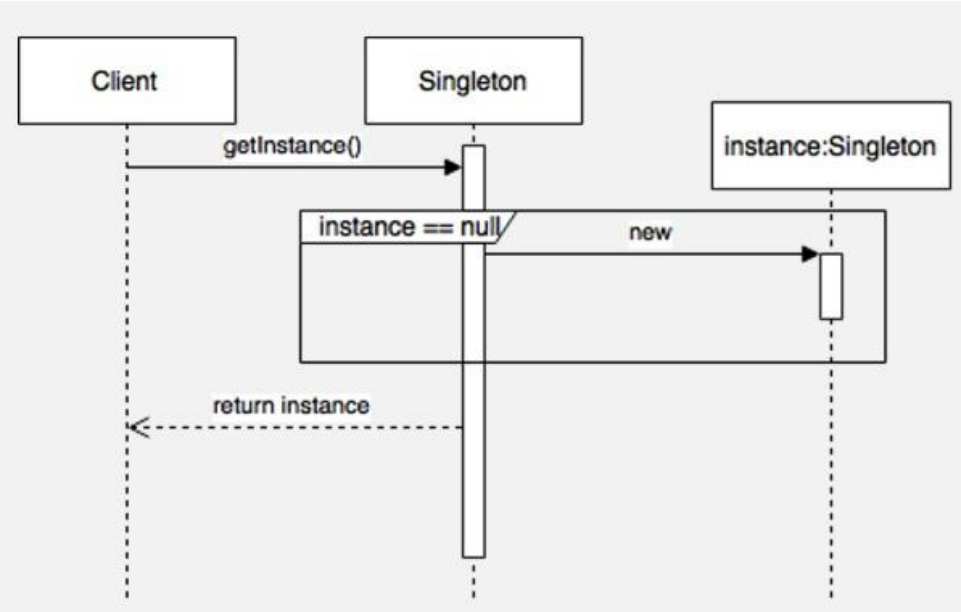


Singleton

```
public class Configuracao {
```

```
    private String nomeUsuario;

    private static Configuracao singleton;

    private Configuracao() {
    }

    public static Configuracao getInstance() {
        if(singleton == null) {
            singleton = new Configuracao();
        }
        return singleton;
    }

    public String getNomeUsuario() {
        return nomeUsuario;
    }

    public void setNomeUsuario(String nomeUsuario) {
        this.nomeUsuario = nomeUsuario;
    }
}
```

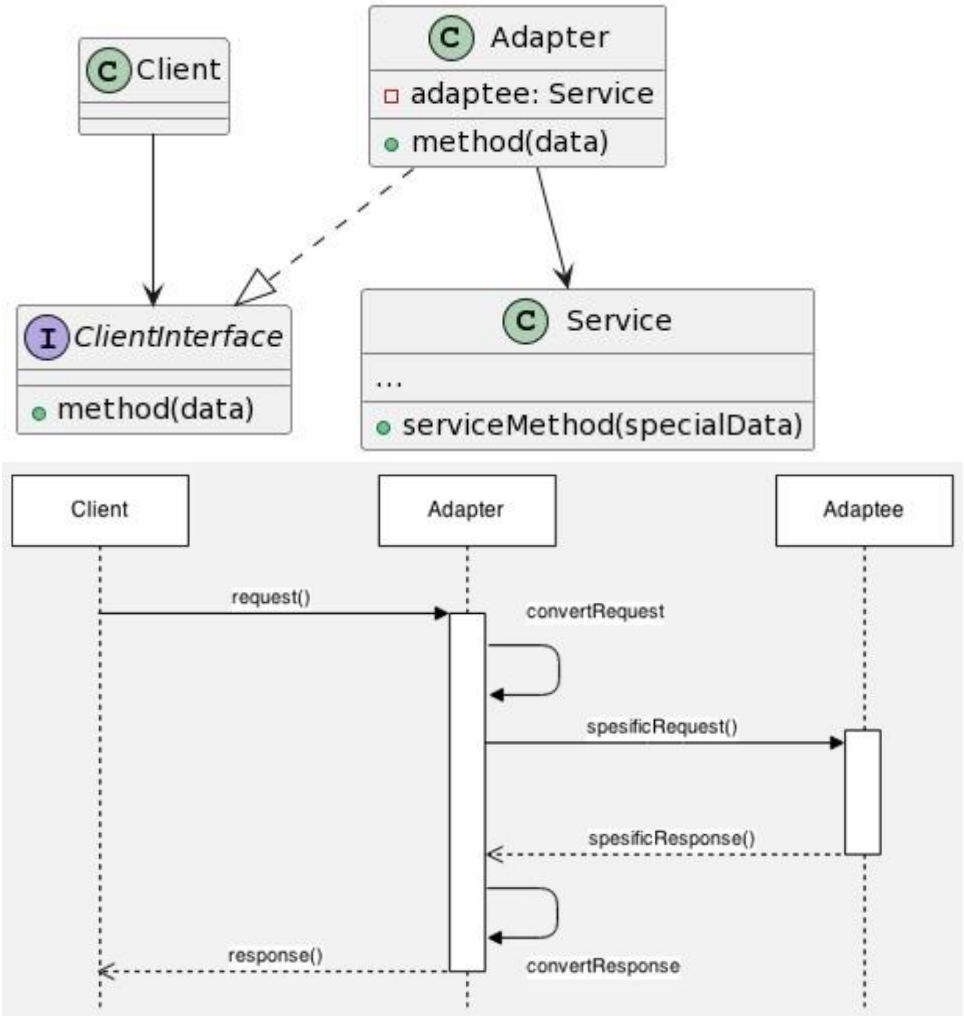




Object Adapter

```
public interface ManipuladorImagem {
public void carregarImagem(String arquivo);
public void desenharImagem(int largura, int altura,
int posicaoX, int posicaoY);
}
public class ManipuladorImagemA extends ManipuladorImagemBibliotecaA
implements ManipuladorImagem {
public void carregarImagem(String arquivo) {
super.carregarImagem(arquivo);
}
public void desenharImagem(int largura, int altura,
int posicaoX, int posicaoY) {
```

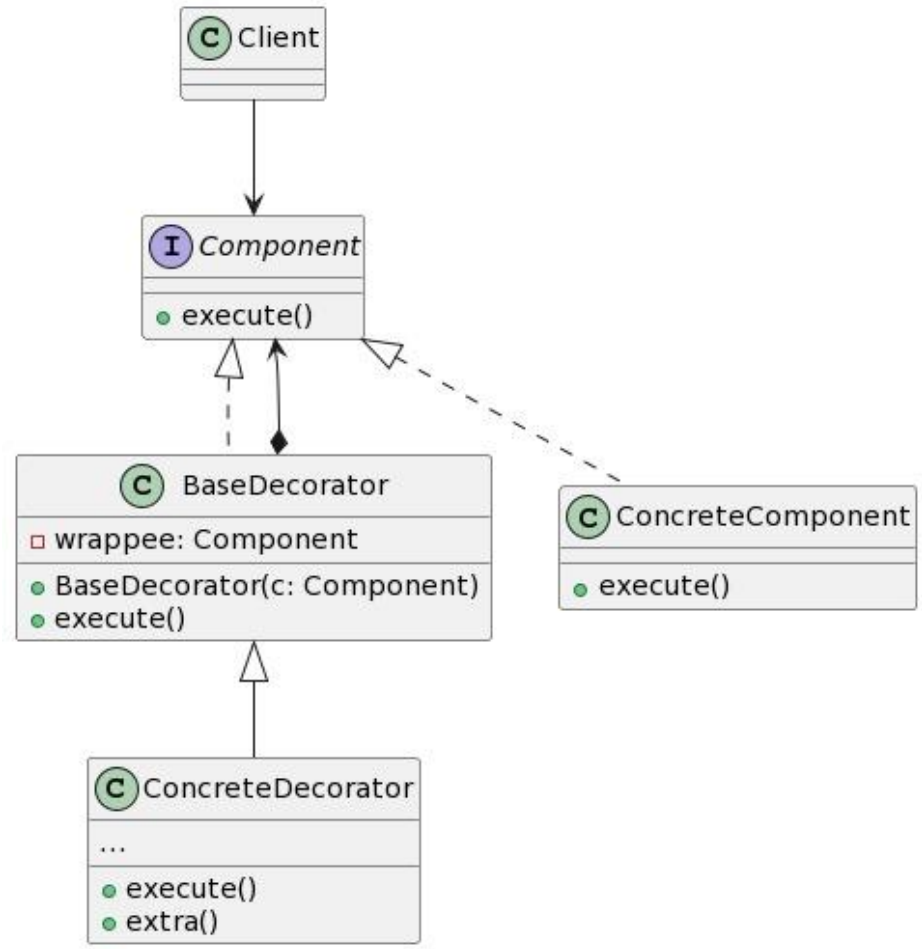
```
super.desenharImagem(posicaoX, posicaoY);
}
}
public static void main(String[] args) {
ManipuladorImagem imagem = new ManipuladorImagemA();
imagem.carregarImagem("teste.png");
imagem.desenharImagem(0, 0, 10, 10);
imagem = new ManipuladorImagemB();
imagem.carregarImagem("teste.png");
imagem.desenharImagem(0, 0, 10, 10);
}
```

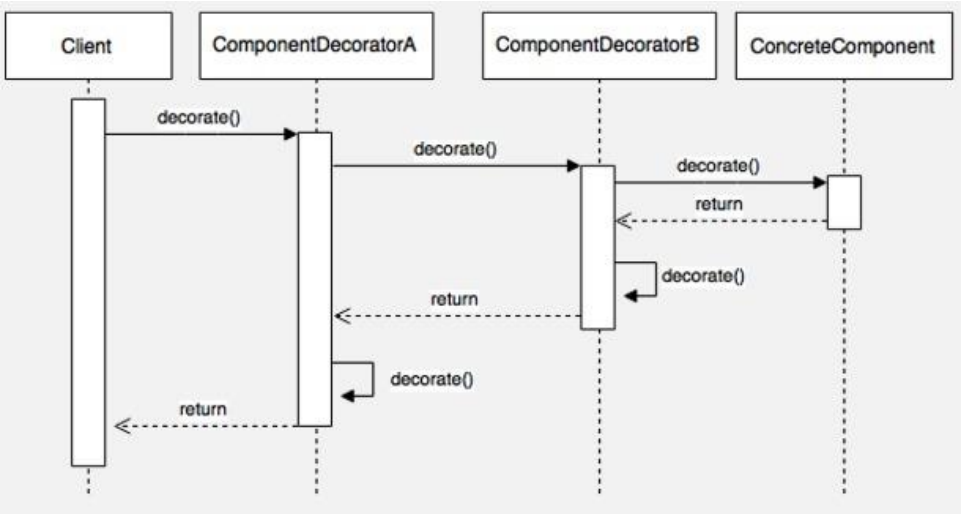


Decorator

```
public interface CalculadoraSalario {

    double calcularSalario(int horasTrabalhadas);

}

public class CalculadoraSalarioFuncionario implements CalculadoraSalario {

    public double calcularSalario(int horasTrabalhadas) {
        final double valorHora = 15d;
        return valorHora * horasTrabalhadas;
    }
}

public abstract class CalculadoraSalarioDecorator implements CalculadoraSalario {

    protected CalculadoraSalario calculadoraSalarioBase;

    public CalculadoraSalarioDecorator(CalculadoraSalario
        calculadoraSalarioBase) {
        this.calculadoraSalarioBase = calculadoraSalarioBase;
    }
}

public class CalculadoraHoraExtraDecorator extends CalculadoraSalarioDecorator {

    public CalculadoraHoraExtraDecorator(CalculadoraSalario
        calculadoraSalarioBase) {
        super(calculadoraSalarioBase);
    }
```

```
    public double calcularSalario(int horasTrabalhadas) {
        double pagamentoHorasExtras = 0d;
        final double valorHoraExtra = 18d;
        if (horasTrabalhadas > 160) {
            int horasExtras = horasTrabalhadas - 160;
            pagamentoHorasExtras = horasExtras * valorHoraExtra;
            return this.calculadoraSalarioBase.calcularSalario(
                horasTrabalhadas - horasExtras) + pagamentoHorasExtras;
        }
        return this.calculadoraSalarioBase.calcularSalario(horasTrabalhadas);
    }
}
```

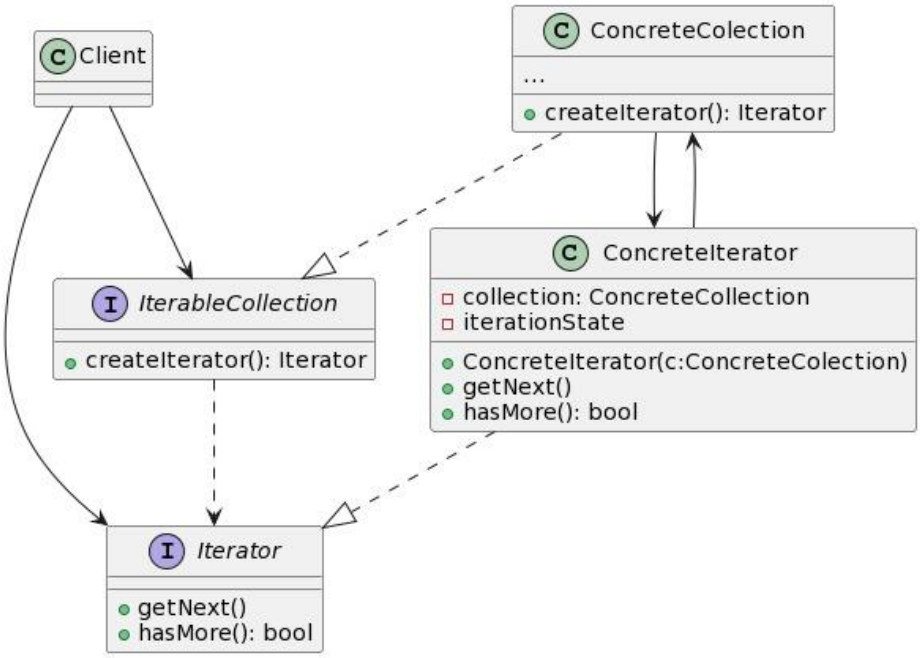




Iterator

```
import java.util.ArrayList;
import java.util.Iterator;

public class Main {
  public static void main(String[] args) {

    // Make a collection
    ArrayList<String> cars = new ArrayList<String>();
```

```
        cars.add("Volvo");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("Mazda");

        // Get the iterator
        Iterator<String> it = cars.iterator();

        // Print the first item
        System.out.println(it.next());
    }
}
```



Observer

botao.addActionListener(new ActionListener() {

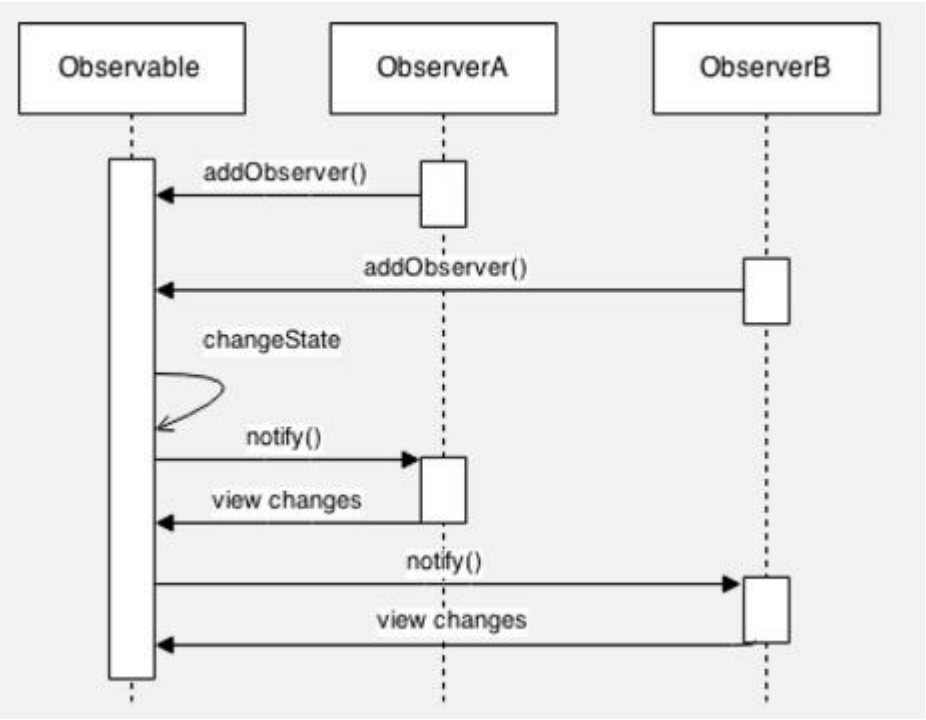
@Override

public void actionPerformed(ActionEvent e) {

System.out.println("Clicado!");

}

})


Observable
ObserverA
ObserverB
addObserver()
addObserver()
changeState
notify()
view changes
notify()
view changes