CAFETEIRA

==========

COMPOSITE

Compõe os sabores de café. Vantagem : ao compor os sabores, é possível somar o preço do sabor com base no preço de cada ingrediente.

```
 */
public class Capuccino extends CafeComposite {

    public Capuccino(String nome) {
        super(nome);
    }

   @Override
    public void preparaReceita() {
        children.add(new CafePuro("CafePuro"));
        children.add(new Leite("Leite"));
        children.add(new Agua("Agua"));
    }

}
```

```java
public abstract class CafeComposite implements TipoCafe {

    protected List<CafeComposite> children = new ArrayList<>();
    protected String nome;

    public CafeComposite(String nome) {
        this.nome = nome;
    }

    public void add(CafeComposite letter) {
        children.add(letter);
    }

    public int count() {
        return children.size();
    }

    public void print() {
        for (CafeComposite letter : children) {
            letter.print();
        }
    }

    @Override
    public float valorTotal() {
        float valor = 0f;
        System.out.println("Para "+nome+" vou adicionar");
        for (CafeComposite ingrediente : children) {
            System.out.println(ingrediente.getNome()+" - R$"+ingrediente.valorTotal());
            valor += ingrediente.valorTotal();
        }

        return valor;
    }
```

FACTORY

Para a escolha do sabor de café. A FACTORY facilita a instanciação do objeto do sabor de acordo com o ENUM

```
public class TipoCafeFactory {
    public static TipoCafe getCapsula(TipoCafeEnum tipo){
        if(tipo == TipoCafeEnum.EXPRESSO){
            return new Expresso("Expresso");
        } else {
            return new Capuccino("Capuccino");
        }
    }
}
```

OBSERVER

Observa os estados da cafeteira, se o copo está cheio, se a mistura dos sabores estão prontas Assim é possível executar regras quando um certo estado for atingido, como quando o copo estiver cheio parar de encher.

```
public class CafeProntoParaEncherCopo implements CafeteiraObserver {

    @Override
    public void update(EstadoCafeteiraEnum estado) {
        if(estado == EstadoCafeteiraEnum.CAFE_PRONTO_PARA_ENCHER){
            System.out.println("Mistura efetuada com sucesso!\n");
        }
    }

}
```

Strategy

Estratégia para os níveis de café no copo. Assim ela pode ter a estratégia de copo baixo ou médio sem alterar as regras de negócio

```
public class NivelAlto  implements CafeteiraStrategy{

    @Override
    public void execute() {
        try {
            System.out.println("enchendo a caneca");
            Thread.sleep(10000);
            System.out.println("caneca cheia\n==============");
        } catch (InterruptedException ex) {
            Logger.getLogger(NivelBaixo.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
```

-