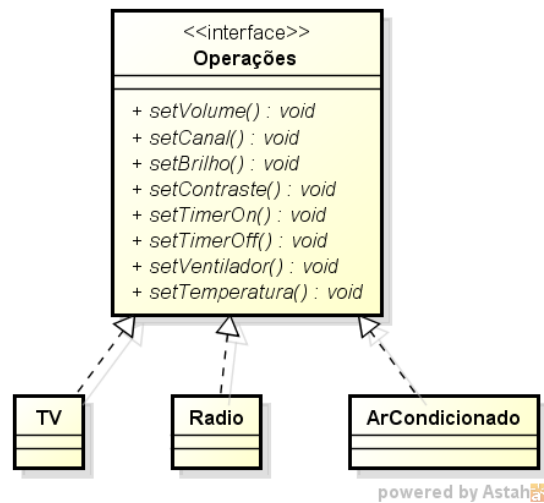




1. Sobre o diagrama abaixo, façam o que se pede.



- O diagrama está coerente com o Princípio da Segregação de Interfaces? Por quê?
- Se julgarem que não está coerente com o princípio, proponham um novo design que seja coerente com esse princípio.

2. Sobre a classe Controlador abaixo, façam o que se pede.

```
public class Controlador
{
    public List<Conta> contas;
    public List<Cliente> clientes;
    public void DepositarQuantiaEmConta(int contaId, double quantia){...}
    public void SacarQuantiaEmConta(long contaId, double quantia){...}
    public Conta ProcurarContaPeloId(long contaId){...}
    public void CriarNovaConta(Cliente cliente){...}
    public void ExcluirConta(long contaId){...}
    public void TransferirQuantia(Conta ctaOrigem, Conta ctaDestino){...}
    public void CadastrarCliente(Cliente cliente){...}
    public void ExcluirCliente(long clienteId){...}
    public void ProcurarClientePeloNome(string nomeCliente){...}
}
```

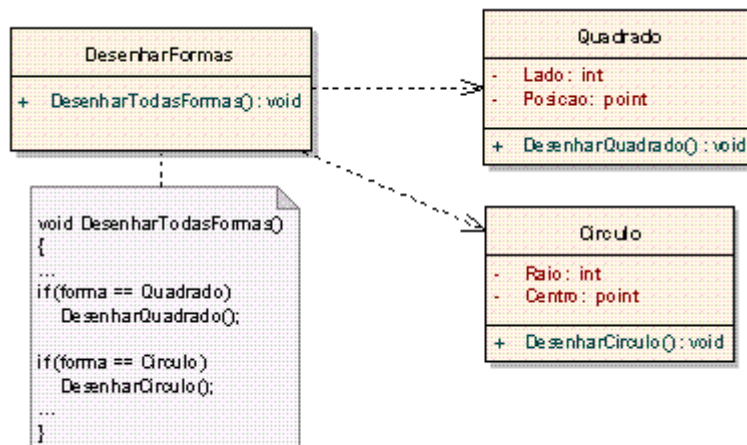
- A classe abaixo está aderente ao Princípio da Responsabilidade Única? Por quê? Indique o que há de errado, se for o caso.
- Se julgarem que não está aderente ao princípio, refatore a classe para deixá-la aderente a esse princípio.

3. Analise a situação abaixo e faça o que se pede a seguir.

```
class Cliente { ... }
class ContaCorrente {
    protected Cliente cliente;
    protected double saldo;
    public double getSaldo() { return saldo; }
    // metodos get e set
    public ContaCorrente ( Cliente c, double saldoInicial ) {
        cliente = c;
        saldo = saldoInicial;
    }
    public void creditar( double valor ) { saldo = saldo + valor; }
    public void debitar( double valor ) {
        if ( valor <= saldo ) saldo = saldo - valor;
    }
}
class ContaEspecial extends ContaCorrente {
    private double limite;
    private double saldoEspecial;
    // metodos get e set
    public ContaEspecial ( Cliente c, double saldoInicial, double
limiteInicial ) {
        super ( c, saldoInicial );
        limite = limiteInicial;
        saldoEspecial = saldoInicial;
    }
    public void debitar( double valor ) {
        if ( valor <= saldo + limite ) {
            if ( valor > saldo ) {
                saldoEspecial = 0;
                limite = limite - (valor - saldo);
            }
            else saldoEspecial = saldo - valor;
        }
    }
}
public class Extrato {
    public static ContaCorrente fabricaContas(
        Cliente c, double saldo, double limite, int tipo ) {
        if (tipo == 1) return new ContaCorrente (c, saldo);
        else return new ContaEspecial (c, saldo, limite);
    }
    public static void main (String args[]) {
        Cliente cliente = new Cliente ();
        ContaCorrente conta = fabricaContas(cliente, 500.00, 200.00, 2);
        System.out.println ( "----- Saldo -----");
        System.out.println ( "saldo = " + conta.getSaldo() );
        conta.debitar ( 100.00 );
        System.out.println ( "----- Saldo -----");
        System.out.println ( "saldo = " + conta.getSaldo() );
    }
}
```

- a) O código está coerente com o Princípio da Substituição de Liskov? Por quê?
- b) Se julgarem que não está coerente com o princípio, reescrevam o que for necessário para que fique coerente com esse princípio.

4. Sobre o diagrama abaixo, façam o que se pede.



- O diagrama está coerente com o Princípio Open/Closed? Por quê?
- Se julgarem que não está coerente com o princípio, proponham um novo design que seja coerente com esse princípio.
- Implemente a nova solução.