1. Indique o ponto onde há a violação do princípio do Bom Cidadão e corrija o código de forma a eliminar o problema (o teste TestaBomCidadao deve passar).

Como dito no enunciado um Titular pode OU NÃO ter dependentes, da forma que a classe Titular estava desenvolvida, não essa regra. Modifiquei o construtor da classe Titular, criando um ArrayList vazio.

```
	public Titular(Integer matricula, String nome) {
		super();
		setMatricula(matricula);
		setNome(nome);
		dependentes = new ArrayList<Dependente>();
	}
```

E criei o método addDependente para poder ter novos dependentes.

```
	public Boolean addDependente(Dependente d1){
		try{
			dependentes.add(d1);
			d1.setTitular(this);
		}
		catch(NullPointerException e){
			return false;
		}

		return true;

	}
```

Obs.: Criei um atributo Titular na classe Dependente

2. Indique a violação do princípio SRP ocorrida no código Titular e faça a refatoração.

A classe Titular viola o princípio SRP ao atribuir pra si, responsabilidade da classe Dependente ao verificar a idade do Dependente. Refatorando ficou como abaixo:

//Classe Titular

```
	public BigDecimal calcularCustoDependentes() {
		BigDecimal total = BigDecimal.ZERO;
		if (dependentes != null) {
			for (Dependente dependente : dependentes) {
				total = total.add(dependente.getValor());
			}
		}
		return total;
	}
```

```
//Classe Dependente
		public BigDecimal getValor() {
		if (getIdade() < 21) {
			valor = JOVEM;
		} else if (getIdade() < 35) {
			valor = MEDIO;
		} else if (getIdade() < 65) {
			valor = ADULTO;
		} else {
			valor = SENIOR;
		}

		return valor;
```

```
	}
```

3. Há a oportunidade para aplicar a refatoração Replace Conditional With Polymorphism. Aplique-a

Criei uma classe abstrata TipoValor que vai servir como uma classe de estado. Implementei os tipos Jovem, Médio, Adulto e Sênior como subclasses dela. Na classe de estado criei um método estático que traz pra si a responsabilidade de checar as idades. No get da classe Dependente chamei esse método estático e limpei a codificação antiga referente.

```
//Classe Dependente
private void setValor() {
		valor = TipoValor.cria(getIdade());
	}
//Classe Tipo Valor – classe de estado
public abstract class TipoValor {

	public static BigDecimal valor;

	abstract public BigDecimal getValor();

	static BigDecimal cria(int idade) {
		if (idade < 21) {
			valor = new Jovem().getValor();
		} else if (idade < 35) {
			valor = new Medio().getValor();
		} else if (idade < 65) {
			valor = new Adulto().getValor();
		} else {
			valor = new Senior().getValor();

		}
		return valor;
	}
}

//Classe Jovem
public class Jovem extends TipoValor {

	@Override
	public BigDecimal getValor() {
		return new BigDecimal("22.00");
	}

}
```

4. Crie ao menos dois testes unitários usando a API Jodatime. Dica: criar testes unitários é a melhor forma de estudar uma API.

Criei diversos testes ao decorrer do desenvolvimento que me ajudaram tanto a entender o uso da API Joda quanto o próprio projeto