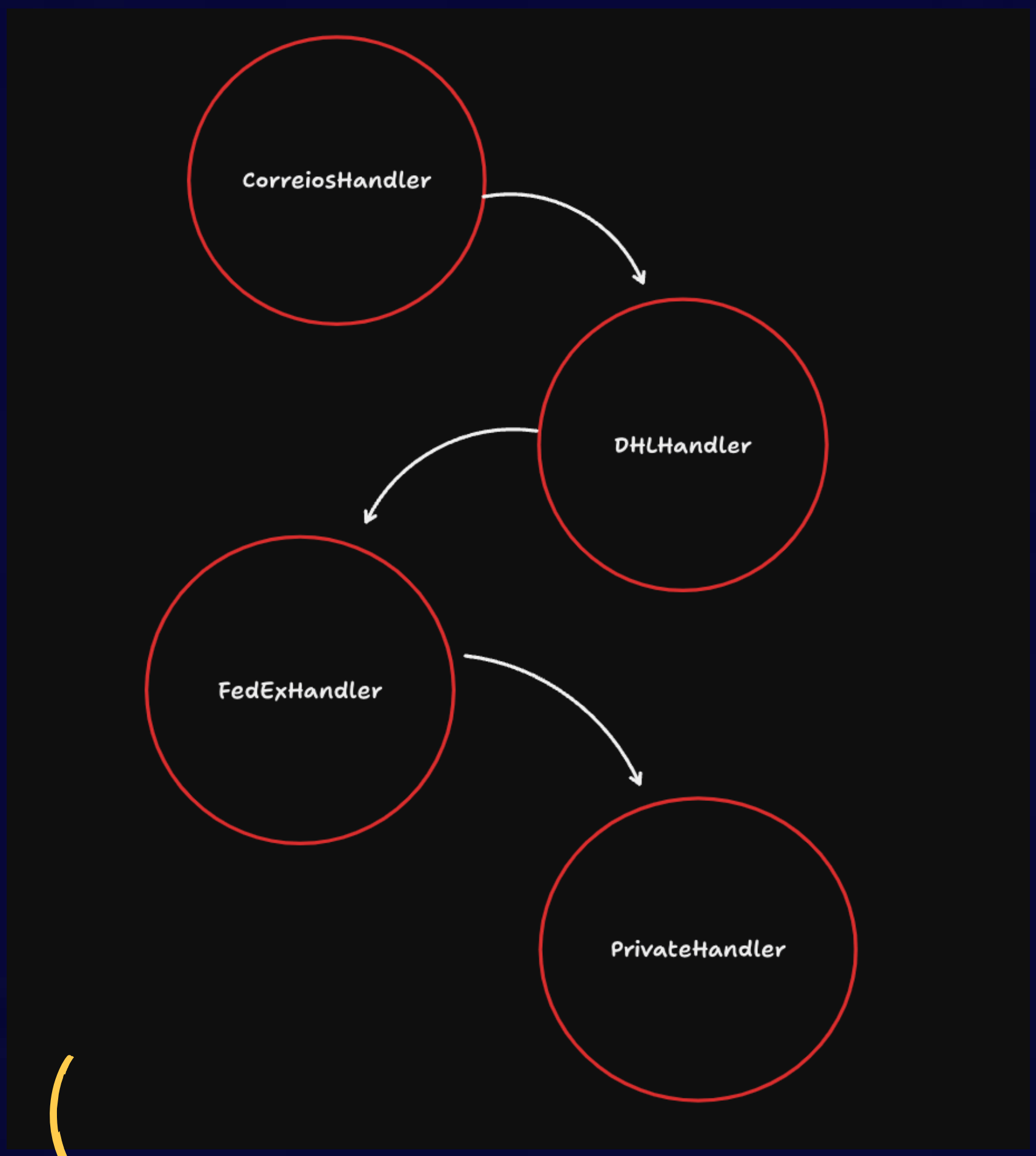
Design Patterns:
Chain of Responsibility



Cadeia de Responsabilidade

Isaac Gomes

TS

Chain of Responsibility

O Chain of Responsibility é um padrão de design comportamental que permite que um pedido seja passado ao longo de uma cadeia de manipuladores (handlers) até que algum deles o processe. Em vez de um único manipulador ser responsável por lidar com o pedido, ele é passado de um objeto para outro até que alguém o processe ou até que a cadeia termine.

Isaac Gomes

TS

Cenário

vamos pensar que estamos calculando uma entrega e temos algumas opções de entrega e seu valor é calculado com base na "transportadora, distancia e período do dia"

Isaac Gomes

Resolução comum

quando pensamos nesse tipo de cenário geralmente pensamos em um Strategy onde temos um factory a instancia correta do Strategy... Mas, vamos resolver com Chain of Responsibility, para simplificar o entendimento vamos pegar uma solução ruim e ir refatorando

Isaac Gomes

```ts
class DeliveryCalculator {
  constructor(
    private carrier: string,
    private timePeriod: string,
    private distance: number
  ) {}

  calculateCost(): number {
    if (this.carrier === 'Correios') {
      if (this.timePeriod === 'morning') {
        return this.distance * 5.5
      } else if (this.timePeriod === 'afternoon') {
        return this.distance * 6.5
      } else {
        return this.distance * 7.5
      }
    } else if (this.carrier === 'DHL') {
      if (this.timePeriod === 'morning') {
        return this.distance * 7.0
      } else if (this.timePeriod === 'afternoon') {
        return this.distance * 8.0
      } else {
        return this.distance * 9.0
      }
    } else if (this.carrier === 'FedEx') {
      if (this.timePeriod === 'morning') {
        return this.distance * 8.5
      } else if (this.timePeriod === 'afternoon') {
        return this.distance * 9.5
      } else {
        return this.distance * 10.5
      }
    } else {
      throw new Error('Unsupported carrier')
    }
  }
}
```

Isaac Gomes

1 - Problema

```
} else if (this.carrier === 'FedEx') {
  if (this.timePeriod === 'morning') {
    return this.distance * 8.5
  } else if (this.timePeriod === 'afternoon') {
    return this.distance * 9.5
  } else {
    return this.distance * 10.5
  }
```

o primeiro problema que notamos são as palavras magicas que não me dão um contexto bom do código e um uso Ruim do TS já que eu tenho meus tipos limitados então não é uma string

Isaac Gomes

1 - Problema

```
constructor(
  private carrier: CarrierType,
  private timePeriod: TimePeriod,
  private distance: number
) {}
```

```
if (this.carrier === CarrierType.Correios) {
  if (this.timePeriod === TimePeriod.Morning) {
    return this.distance * 5.5
  } else if (this.timePeriod === TimePeriod.Afternoon
    return this.distance * 6.5
  } else {
    return this.distance * 7.5
  }
```

então vamos usar um enum para limitar os tipos e eliminar as palavras magicas do nosso código

Isaac Gomes

TS

2 - Problema

```
if (this.carrier === CarrierType.Correios) {
  if (this.timePeriod === TimePeriod.Morning) {
    return this.distance * 5.5
  } else if (this.timePeriod === TimePeriod.Afternoon) {
    return this.distance * 6.5
  } else {
    return this.distance * 7.5
  }
} else if (this.carrier === CarrierType.DHL) {
  if (this.timePeriod === TimePeriod.Morning) {
    return this.distance * 7.0
  } else if (this.timePeriod === TimePeriod.Afternoon) {
    return this.distance * 8.0
  } else {
    return this.distance * 9.0
  }
```

note que temos periodos
mapeados e mesmo assim
repetimos a mesma tratativa
podemos centralizar essa tratativa

Isaac Gomes

TS

2 - Problema

```
class RateFactory {
  static createRates(parmas: TimePeriodRates) {
    const {
      afternoonRate,
      eveningRate,
      morningRate
    } = parmas
    return {
      [TimePeriod.Morning]: morningRate,
      [TimePeriod.Afternoon]: afternoonRate,
      [TimePeriod.Evening]: eveningRate,
    }
  }
}
```

então vamos criar um Factory para cuidar desse valor por periodo, vamos usar um objeto literal do tipo Record<TimePeriod, number>

Isaac Gomes

```ts
export class DeliveryCalculator {
  constructor(
    private carrier: CarrierType,
    private timePeriod: TimePeriod,
    private distance: number
  ) {}

  calculateCost(): number {
    if (this.carrier === CarrierType.Correios) {
      const correiosRates = TimePeriodPricing.createRates({
        morningRate: 5.5,
        afternoonRate: 6.5,
        eveningRate: 7.5,
      })
      return correiosRates[this.timePeriod] * this.distance
    } else if (this.carrier === CarrierType.DHL) {
      const dhlRates = TimePeriodPricing.createRates({
        morningRate: 7.0,
        afternoonRate: 8.0,
        eveningRate: 9.0,
      })
      return dhlRates[this.timePeriod] * this.distance
    } else if (this.carrier === CarrierType.FedEx) {
      const fedexRates = TimePeriodPricing.createRates({
        morningRate: 8.5,
        afternoonRate: 9.5,
        eveningRate: 10.5,
      })
      return fedexRates[this.timePeriod] * this.distance
    } else {
      throw new Error('Unsupported carrier')
    }
  }
}
```

Isaac Gomes

3 - Problema

```
if (this.carrier === CarrierType.Correios) {
  const correiosRates = TimePeriodPricing.createRates({
    morningRate: 5.5,
    afternoonRate: 6.5,
    eveningRate: 7.5,
  })
  return correiosRates[this.timePeriod] * this.distance
} else if (this.carrier === CarrierType.DHL) {
  const dhlRates = TimePeriodPricing.createRates({
    morningRate: 7.0,
    afternoonRate: 8.0,
    eveningRate: 9.0,
  })
```

Concorda que ele nunca vai usar duas transportadora para mesma entrega? então podemos remover esses ifs usando uma cadeia de Handler... Assim, mantemos o open/closed

Isaac Gomes

TS

3 - Problema

```
interface ShippingCalculatorHandler {
  calculateCost(shippingRequest: ShippingRequest): number
}
```

*cada **if identifica** se é seu tipo se não for vai para o outro if e se for retorna o calculo... então teremos um metodo **calculateCost** em cada **handler***

Isaac Gomes

TS

3 - Problema

```
class CorreiosHandler implements ShippingCalculatorHandler {
  constructor(
    private next?: ShippingCalculatorHandler,
    private ratesCorreios = RateFactory.createRates({
      morningRate: 5.5,
      afternoonRate: 6.5,
      eveningRate: 7.5,
    })
  ) {}

  calculateCost(requestData: ShippingRequest): number {
    const { carrier, distance, timePeriod } = requestData
    const isCorreiosCarrier = carrier === CarrierType.Correios
    const totalShippingCost = this.ratesCorreios[timePeriod] * distance
    if (isCorreiosCarrier) return totalShippingCost
    if (!this.next) throw new UnsupportedCarrierError(carrier)
    return this.next.calculateCost(requestData)
  }
}
```

agora vamos implementar o nosso handler ele cria seus valores por periodo, verifica se é ele que trata se for retorna o valor senão chama o proximo handler

Isaac Gomes

Resultado

```
class DeliveryCalculator {
  calculateCost(requestData: ShippingRequest) {
    const correiosHandler = new CorreiosHandler()
    const dhlHandler = new DHLHandler(correiosHandler)
    const fedExHandler = new FedExHandler(dhlHandler)
    return fedExHandler.calculateCost(requestData)
  }
}
```

com isso vamos ter um fluxo de handler
o fedExHandler trata? Não
chama o dhlHandler
o dhlHandler trata? Não
chama o correiosHandler
o correiosHandler trata? Sim
retorna o valor

Isaac Gomes

TS

saimos disso

```
class DeliveryCalculator {
  constructor(
    private carrier: string,
    private timePeriod: string,
    private distance: number
  ) {}

  calculateCost(): number {
    if (this.carrier === 'Correios') {
      if (this.timePeriod === 'morning') {
        return this.distance * 5.5
      } else if (this.timePeriod === 'afternoon') {
        return this.distance * 6.5
      } else {
        return this.distance * 7.5
      }
    } else if (this.carrier === 'DHL') {
      if (this.timePeriod === 'morning') {
        return this.distance * 7.0
      } else if (this.timePeriod === 'afternoon') {
        return this.distance * 8.0
      } else {
        return this.distance * 9.0
      }
    } else if (this.carrier === 'FedEx') {
      if (this.timePeriod === 'morning') {
        return this.distance * 8.5
      } else if (this.timePeriod === 'afternoon') {
        return this.distance * 9.5
      } else {
        return this.distance * 10.5
      }
    } else {
      throw new Error('Unsupported carrier')
    }
  }
}
```

Isaac Gomes

TS

para isso

```
class DeliveryCalculator {
  calculateCost(requestData: ShippingRequest) {
    const correiosHandler = new CorreiosHandler()
    const dhlHandler = new DHLHandler(correiosHandler)
    const fedExHandler = new FedExHandler(dhlHandler)
    return fedExHandler.calculateCost(requestData)
  }
}
```

Onde temos melhor uso do TS, Divisão das responsabilidades, Chain of Responsibility e uma implementação que usa Open/Closed, agora se quisermos adicionar é só seguir o padrão dos handler

Isaac Gomes

TS

vamos adicionar mais uma **transportadora** para evidenciar

```
class UPSHandler implements ShippingCalculatorHandler {
  constructor(
    private next?: ShippingCalculatorHandler,
    private ratesUPS = RateFactory.createRates({
      morningRate: 6.0,
      afternoonRate: 7.0,
      eveningRate: 8.0,
    })
  ) {}

  calculateCost(requestData: ShippingRequest): number {
    const { carrier, distance, timePeriod } = requestData
    const isUPSCarrier = carrier === CarrierType.UPS
    const totalShippingCost = this.ratesUPS[timePeriod] * distance
    if (isUPSCarrier) return totalShippingCost
    if (!this.next) throw new UnsupportedCarrierError(carrier)
    return this.next.calculateCost(requestData)
  }
}
```

Isaac Gomes

TS

RESULTADO

```
class DeliveryCalculator {
  calculateCost(requestData: ShippingRequest) {
    const correiosHandler = new CorreiosHandler()
    const dhlHandler = new DHLHandler(correiosHandler)
    const fedExHandler = new FedExHandler(dhlHandler)
    const upsHandler = new UPSHandler(fedExHandler)
    return upsHandler.calculateCost(requestData)
  }
}
```

pronto é só chamar na nossa cadeia de handler e pronto...

Isaac Gomes

TS

Gostou?

Curta

Compartilhe

Salve

Isaac Gomes