Pergunta 1

1. O que acontece se extrato de um cliente deve agora retornar no formato HTML, ao invés de String?

Para isso, foi utilizado uma classe abstrata `ExtratoFormatter`, que será usada para definir como será o retorno.

```
from abc import ABC, abstractmethod
from rent import Rent

# Interface para diferentes estratégias de formatação
class ExtratoFormatter(ABC):
    @abstractmethod
    def formatar_extrato(self, nome: str, alugueis: list[Rent], valor_total:
float, pontos: int) -> str:
        pass

# Implementação de extrato como string
class ExtratoTextoFormatter(ExtratoFormatter):
    def formatar_extrato(self, nome: str, alugueis: list[Rent], valor_total:
float, pontos: int) -> str:
        fim_de_linha = "\n"
        resultado = f"Registro de Alugueis de {nome}{fim_de_linha}"

        for rent in alugueis:
            valor_corrente = rent.get_custo_aluguel()
            resultado += f"\t{rent.get_tape().get_titulo()}\t{valor_corrente}
{fim_de_linha}"

        resultado += f"Valor total devido: {valor_total}{fim_de_linha}"
        resultado += f"Você acumulou {pontos} pontos de alugador frequente"
        return resultado

# Implementação de extrato como HTML
class ExtratoHTMLFormatter(ExtratoFormatter):
    def formatar_extrato(self, nome: str, alugueis: list[Rent], valor_total:
float, pontos: int) -> str:
        resultado = f"<h1>Registro de Alugueis de {nome}</h1><ul>"

        for rent in alugueis:
            valor_corrente = rent.get_custo_aluguel()
            resultado += f"<li>{rent.get_tape().get_titulo()}: {valor_corrente}
</li>"

        resultado += f"</ul><p>Valor total devido: {valor_total}</p>"
        resultado += f"<p>Você acumulou {pontos} pontos de alugador frequente</p>"
        return resultado
```

A classe `Extrato` utiliza de objetos `CalculadoraDeExtrato` e `ExtratoFormatter`, ambos sendo classes abstratas, seguindo o DIP.

```
from rent import Rent
from calculadora_extrato import CalculadoraDeExtrato
from extrato_formatter import ExtratoFormatter

class Extrato:
    def __init__(self, nome: str, alugueis: list[Rent], calculadora_extrato:
CalculadoraDeExtrato, extrato_formatter: ExtratoFormatter):
        self.nome: str = nome
        self.alugueis: list[Rent] = alugueis
        self.calculadora_extrato: CalculadoraDeExtrato = calculadora_extrato
        self.extrato_formatter: ExtratoFormatter = extrato_formatter

    def gerar_extrato(self) -> str:
        valor_total = self.calculadora_extrato.calcular_valor_total()
        pontos_de_alugador_frequente =
self.calculadora_extrato.calcular_pontos_totais()
        return self.extrato_formatter.formatar_extrato(self.nome, self.alugueis,
valor_total, pontos_de_alugador_frequente)
```

Classe `CalculadoraDeExtrato`:

```
from rent import Rent

class CalculadoraDeExtrato:
    def __init__(self, alugueis: list[Rent]):
        self.alugueis = alugueis

    def calcular_valor_total(self) -> float:
        valor_total = 0.0
        for rent in self.alugueis:
            valor_total += rent.get_custo_aluguel()
        return valor_total

    def calcular_pontos_totais(self) -> int:
        pontos_total = 0
        for rent in self.alugueis:
            pontos_total += rent.get_pontos_de_alugador()
        return pontos_total
```

Assim, o código de `Client` ficou desta maneira:

```
from extrato import Extrato
from rent import Rent
from calculadora_extrato import CalculadoraDeExtrato
from extrato_formatter import ExtratoTextoFormatter, ExtratoHTMLFormatter

class Client:
    def __init__(self, nome):
```

```
        self.nome = nome
        self.tapes_alugadas = []

    def get_nome(self):
        return self.nome

    def adiciona_rent(self, rent):
        self.tapes_alugadas.append(rent)

    def gerar_extrato_html(self):
        extrato = Extrato(self.nome, self.tapes_alugadas, CalculadoraDeExtrato(
            self.tapes_alugadas), ExtratoHTMLFormatter())

        html_content = f"""
            <!DOCTYPE html>
            <html lang="pt-BR">
            <head>
                <meta charset="UTF-8">
                <meta name="viewport" content="width=device-width, initial-
scale=1.0">
                <title>Página Gerada em Python</title>
            </head>
            <body>
                {extrato.gerar_extrato()}
            </body>
            </html>
            """

        with open("extrato.html", "w", encoding="utf-8") as file:
                file.write(html_content)

        return extrato.gerar_extrato()

    def gerar_extrato_text(self):
        extrato = Extrato(self.nome, self.tapes_alugadas, CalculadoraDeExtrato(
            self.tapes_alugadas), ExtratoTextoFormatter())
        return extrato.gerar_extrato()
```

Ao definir que o retorno será do tipo HTML, um arquivo .html será cria, contendo as informações referentes para o extrato.

Pergunta 2

2. O que ocorre se as regras de cálculo e preço mudarem?

Outra vez foi utilizado de classe abstrata, seguindo o DIP, para resolver esse problema.

```
from abc import ABC, abstractmethod

class ValorCalculo(ABC):
        @abstractmethod
```

```
        def calcular(self, dias_alugada: int):
                pass

class NormalCalculo(ValorCalculo):
        def calcular(self, dias_alugada: int):
                valor = 2
                if dias_alugada > 2:
                        valor += (dias_alugada - 2) * 1.5
                return valor

class LancamentoCalculo(ValorCalculo):
        def calcular(self, dias_alugada: int):
                return dias_alugada * 3

class InfantilCalculo(ValorCalculo):
        def calcular(self, dias_alugada: int):
                valor = 1.5
                if dias_alugada > 3:
                        valor += (dias_alugada - 3) * 1.5
                return valor
```

Desse jeito, é possível alterar individualmente cada cálculo de categoria.

Com essa classe, podemos realizar o custo do aluguel para cada fita, usando o método `calcular_valor_aluguel` da classe `Tape`.

```
from valor_calculo import ValorCalculo, NormalCalculo, InfantilCalculo,
LancamentoCalculo
from calculadora_pontos import CalculadoraDePontos, CalculadoraDePontosNormal,
CalculadoraDePontosLancamento, CalculadoraDePontosInfantil
from enum import Enum
from abc import ABC

class TapeType(Enum):
    NORMAL = 0
    LANCAMENTO = 1
    INFANTIL = 2

class GerenciadorDeTapes():
    def __init__(self,
                 type: TapeType = TapeType.NORMAL,
                 calculadora_pontos:CalculadoraDePontos =
CalculadoraDePontosNormal,
                 calculadora_aluguel: ValorCalculo = NormalCalculo
                 ) -> None:
        self._calculadora_pontos = calculadora_pontos
        self._calculadora_aluguel = calculadora_aluguel

    def calcular_pontos(self, dias_alugada:int):
        return self._calculadora_pontos.calcular(dias_alugada)

    def calcular_custo_aluguel(self, dias_alugada: int):
```

```
        return self._calculadora_aluguel.calcular(dias_alugada)

class GerenciadorDeTapesNormal(GerenciadorDeTapes):
    def __init__(self) -> None:
        super().__init__(type=TapeType.NORMAL,
calculadora_pontos=CalculadoraDePontosNormal(),
calculadora_aluguel=NormalCalculo())

class GerenciadorDeTapesLancamento(GerenciadorDeTapes):
    def __init__(self) -> None:
        super().__init__(type=TapeType.LANCAMENTO,
calculadora_pontos=CalculadoraDePontosLancamento(),
calculadora_aluguel=LancamentoCalculo())

class GerenciadorDeTapesInfantil(GerenciadorDeTapes):
    def __init__(self) -> None:
        super().__init__(type=TapeType.INFANTIL,
calculadora_pontos=CalculadoraDePontosInfantil(),
calculadora_aluguel=InfantilCalculo())

class Tape:
    def __init__(self, titulo: str, gerenciadorDeTapes:GerenciadorDeTapes) ->
None:
        self.titulo: str = titulo
        self._gerenciadorDeTapes = gerenciadorDeTapes

    def get_titulo(self) -> str:
        return self.titulo

    def mudarTipo(self, novoGerenciamento:GerenciadorDeTapes):
        self._gerenciadorDeTapes = novoGerenciamento

    def calcular_custo_aluguel(self, numeroDias:int) -> int:
        return self._gerenciadorDeTapes.calcular_custo_aluguel(numeroDias)

    def calcular_pontos(self, numeroDias: int) -> int:
        return self._gerenciadorDeTapes.calcular_pontos(numeroDias)
```

Foi criado um classe `GerenciadorDeTapes` que será responsável por realizar os cálculos relacionados aos custos e pontos.

Também sendo possível usar o método `get_custo_alugel` da classe `Rent`.

```
from tape import Tape

class Rent:
        def __init__(self, tape: Tape, dias_alugada: int):
                self.tape: Tape = tape
                self.dias_alugada: int = dias_alugada

        def get_tape(self) -> Tape:
                return self.tape
```

```
        def get_dias_alugada(self) -> int:
                return self.dias_alugada

        def get_custo_aluguel(self):
                return self.tape.calcular_valor_aluguel(self.dias_alugada)

        def get_pontos_de_alugador(self):
                return self.tape.calcular_pontos(self.dias_alugada)
```

Pergunta 3

3. Se classificação das fitas mudar toda semana?

Para isso, seria necessário modificar (adicionar, remover, editar) as classes que herdam de `ValorCalculo` e de `CalculadoraDePontos`.

Classe `CalculadoraDePontos`:

```
from abc import ABC, abstractmethod

class CalculadoraDePontos(ABC):
        @abstractmethod
        def calcular(self, dias_alugada: int):
                pass

class CalculadoraDePontosNormal(CalculadoraDePontos):
        def calcular(self, dias_alugada: int):
                return 1

class CalculadoraDePontosLancamento(CalculadoraDePontos):
        def calcular(self, dias_alugada: int):
                pontos = 1
                if dias_alugada > 1:
                        pontos += 1
                return pontos

class CalculadoraDePontosInfantil(CalculadoraDePontos):
        def calcular(self, dias_alugada: int):
                return 1
```

Pergunta 4

4. Se esquema de pontos de alugador puder mudar a qualquer hora?

Seria necessário mudar apenas as classes que herdam de `CalculadoraDePontos`.

Uma vez que a criação do objeto `Tape` depende tanto de `ValorCalculo`, como de `CalculadoraDePontos`.