**Verificação e Validação de Software**

**Prof. Bernardo Copstein**

**Trabalho final**

# Teste do sistema GarOOPa – Versão Simplificada

## Descrição geral

**GarOOPa** é um aplicativo de mobilidade urbana através do qual o passageiro, previamente cadastrado, pode solicitar um veículo – de uma determinada categoria – para realizar um deslocamento mediante pagamento.

Nesta versão simplificada o sistema atende a uma única cidade cujos bairros estão cadastrados. Cada bairro é definido por uma região retangular definida por dois pontos (X1,Y1- X2,Y2) e possui um nome e um custo associados. Quando um cliente solicita uma viagem de um bairro até outro o sistema define uma rota. A rota corresponde a uma reta partindo do ponto central do bairro origem até o ponto central do bairro destino conforme a figura:

|  |
| --- |
|  |

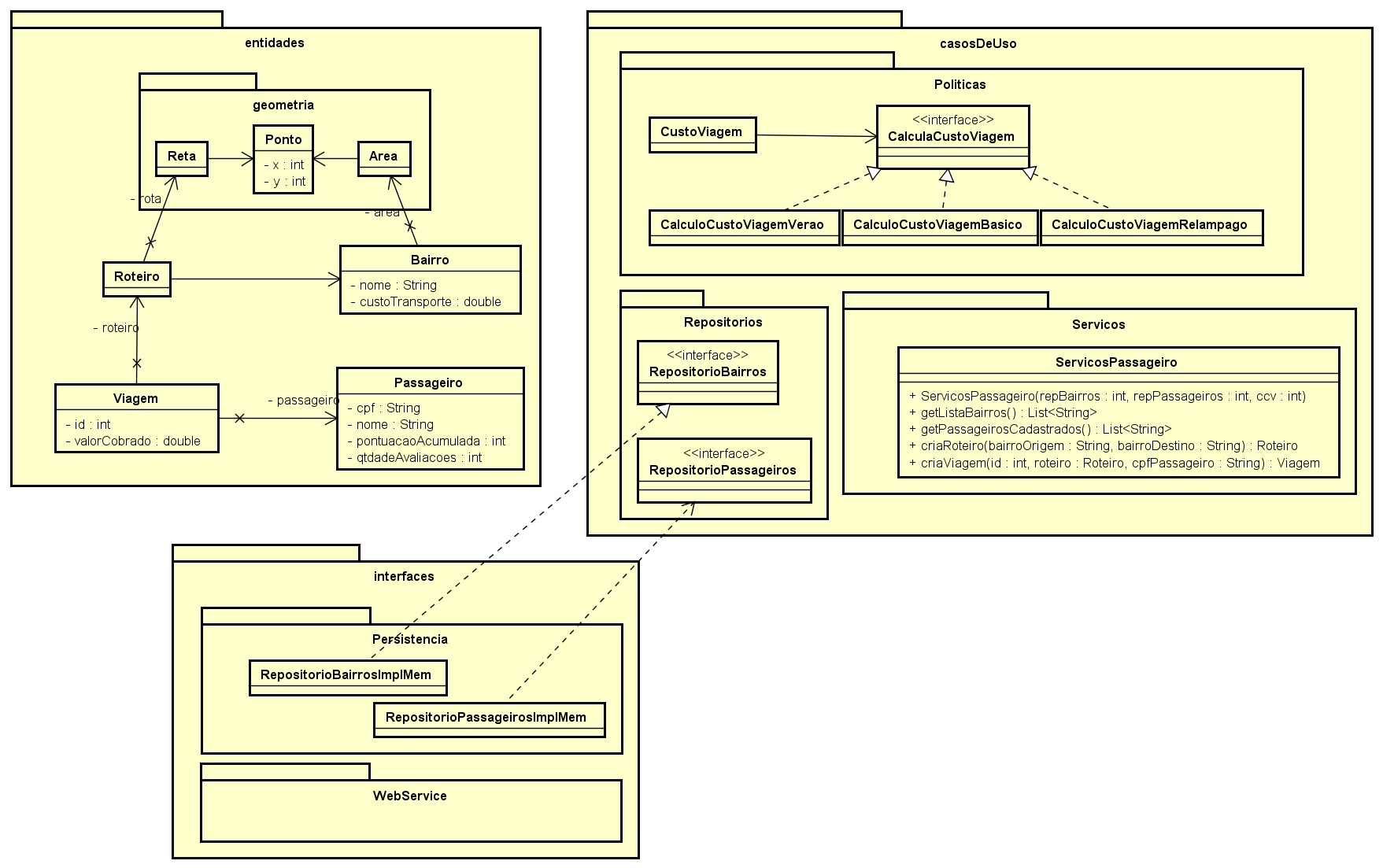
Figura 1: exemplo de mapa de cidade e rota

O custo básico do deslocamento corresponde ao somatório dos custos associados aos bairros interseccionados pela rota. Se o deslocamento for dentro do mesmo bairro então o custo básico corresponde apenas ao custo associado ao bairro. No caso do exemplo da figura o custo do deslocamento de Petrópolis até Menino Deus seria de R$ 39,00 (10+12+8+9) visto que a rota intersecciona 4 bairros (Petrópolis, Itá, Bom Fim e Menino Deus).

Os passageiros que solicitam as viagens devem estar previamente cadastrados no sistema. Sobre cada viagem anota-se um identificador, a data/hora do início, o passageiro, o roteiro e o valor cobrado. Ao final da viagem o passageiro é avaliado pelo motorista. Passageiros bem avaliados podem obter descontos em viagens futuras. Todo passageiro ingressa no sistema com nota 8, podendo melhor ou piorar seu escore à medida que vai sendo avaliado. Um roteiro é composto pelos bairros origem e destino da viagem, bem como pela lista de bairros que a viagem atravessa (definidos automaticamente pelo sistema). O cálculo do custo da viagem depende da política vigente, porém sempre usa como parâmetros de entrada o roteiro e os dados do passageiro.

## As classes

O diagrama apresenta o conjunto de classes fornecidos organizados de acordo com a arquitetura CLEAN. No nível de entidades temos as classes do pacote “Geometria” responsáveis pelas primitivas gráficas básicas (ponto, reta e área). Na classe “Area” está localizado um dos algoritmos mais importantes: o que permite definir se uma reta corta um retângulo ou não. Esse algoritmo é fundamental para definir os bairros que um roteiro percorre.



As demais classes do nível “Entidades” descrevem as demais entidades com as quais o sistema lida: bairro, roteiro, viagem e passageiro. No nível de interface foram disponibilizadas apenas as classes que implementam versões em memória dos repositórios de dados. Já no nível de casos de uso temos a classe responsável pelo cálculo do valor a ser cobrado pelas viagens de acordo com a política vigente e a fachada de serviços que implementa as funcionalidades do sistema, a saber: obter a lista de bairros, obter a lista de passageiros, criar um roteiro e criar uma viagem (o usuário define um roteiro indicando os bairros de origem e destino – o sistema calcula que bairros são percorridos – e então solicita a viagem. Neste momento é que o custo é calculado).

Em relação ao custo das viagens a política básica não prevê descontos por pontuação do cliente ou desconto sazonal. Na promoção relâmpago cliente com mais de 30 avaliações e pontuação média maior que 5 tem 5% de desconto. Além disso qualquer cliente tem 5% de desconto se a viagem percorrer mais de 3 bairros (os descontos são cumulativos). Já na promoção verão clientes com pontuação média acima de 9.0 tem 9% de desconto. Além disso qualquer viagem que percorra mais de 2 bairros tem 10% de desconto (os descontos são cumulativos).

## Objetivos do trabalho

O objetivo do trabalho é identificar defeitos a partir de testes unitários e de integração além da análise estática de código usando o SonarCloud. Na sequência são indicadas as classes para as quais deve-se prever teste unitário (explorando dubles quando necessário) e depois aquelas para as quais deve-se prever testes de integração. Depois de criados todos os drivers de teste, é necessário submeter o código no SonarCloud para verificar todos os “problemas” e a cobertura do código pelos testes. Se for necessário, novos testes devem ser projetados para garantir a cobertura do código.

O foco do trabalho está na geração dos casos de teste, definição da ordem de integração, construção e execução dos drivers e geração dos relatórios. O reparo dos defeitos só deve ser feito quanto se tratar de questões simples de resolver.

## Artefatos a serem entregues e apresentação final

Deverá ser entregue um arquivo .zip contendo toda a estrutura do projeto com suas classes e drivers de teste. Além disso deverá ser elaborada uma apresentação de slides explicando como foram gerados os casos de teste, detalhes significativos dos drivers gerados, como foi planejada a ordem de integração, os resultados obtidos os testes e os novos casos gerados se for o caso. Além da entrega dos artefatos o trabalho deverá ser apresentado para o professor através do encontro síncrono no software Zoom na data definida no Moodle. Se o trabalho for entregue, mas não apresentado, haverá um desconto de 30% na nota. Problemas de conexão ou dificuldade de acesso na data de apresentação devem ser conversados com o professor. Se for de interesse do grupo a apresentação pode ser antecipada desde que previamente combinado com o professor.

## Observações

Espera-se que o trabalho seja desenvolvido ao longo do mês de junho com o acompanhamento do professor. Para tanto é necessário que pelo menos uma vez na semana os grupos se façam presentes nos encontros síncronos para que o professor possa acompanhar e orientar o desenvolvimento do trabalho. Pelo menos 3 defeitos significativos foram “implantados” no código fornecido.

## Classes a serem testadas

|  |  |
| --- | --- |
| **Nível** | **Classes** |
| Entidades | Area, Bairro, Roteiro, Passageiro, Viagem |
| Casos de uso | Toda as classes desse nível devem ser testadas |
| Interface | Nenhuma classe desse nível necessita ser testada (para este trabalho) |