UAlg FCT
UNIVERSIDADE DO ALGARVE
FACULDADE DE CIÊNCIAS E TECNOLOGIA

Licenciatura em Engenharia Informática

Unidade Curricular de
Programação Orientado a Objetos
(2º ano/2semestre – 2019 / 2020)

Relatório do Problema G do Grupo 4

Docente: José Valente de Oliveira

Discente:

David Fernandes, nº:58604

Guilherme Correia, nº:61098

Bruno Susana, nº:61024

Índice

Introdução

Serve o presente relatório para expor todo o trabalho realizado pelo grupo relativo ao lab4, para a resolução deste exercício foi necessário ter em atenção vários conhecimentos matemáticos, dos quais, representação de pontos no plano, noção de distância entre dois pontos, a noção de mediatriz e a de circunferência.

O objetivo foi implementar um algoritmo que dados três pontos distintos pertencentes a uma circunferência e um ponto que garantidamente está fora dessa mesma circunferência, primeiro calculando com os pontos da circunferência se calcule o centro da mesma e de seguida a distância entre esse ponto central e o ponto exterior.

Metodologia

Um dos maiores dilemas deste trabalho foi a escolha do construtor da classe Circunferencia, pois poderíamos ter um HashMap<SegementoReta,double >,e ao iniciar o construtor percorreríamos o array de segmentos, de modo a encontrar as 2 distancias menores. Para depois encontramos as retas perpendiculares a esses 2 segmentos.

As vantagens deste método é que permite facilmente calcular o centro da circunferência.

A nossa decisão foi tomada assim devido as restrições que o problema mete. Tal como é descrito no Anexo A.

Material

-javadoc

-java

-visual paradigm

-eclipse/visual studio

UML

UML é a linguagem standard para especificar, visualizar, construir e documentar os artefactos dos sistemas de software. UML foi criada pelo "Object Management Group" (OMG), tendo sido proposto o rascunho da especificação do UML 1.0 em Janeiro de 1997. UML é uma linguagem orientada a objectos (OO) .

A OO olha para o mundo em termos de classes, relações e objetos, e baseia-se nos seguintes pressupostos:

- As classes são tipos de entidade que agrupam objetos com características semelhantes
- As características das classes são definidas através dos seus atributos que possuem e/ou das operações que realizam

- As classes têm tipos de relações com outras classes.
- Os objetos são indivíduos que pertencem a uma dada classe.
- Os objetos têm ligações com outros objetos determinadas pelos tipos de relação que existem entre as suas respetivas classes

A tabela em baixo sumariza os princípios fundamentais da OO:[1]

Princípio	Descrição	Exemplo
Abstração	Descreve a essência dum objecto (i.e.deixa fora os detalhes) para um dado propósito	Um diagrama do circuito dum filtro de ar descreve as conexões essenciais de forma a que quem mexe nele não eletrocute
Encapsulamento	Descreve apenas o quê é preciso para USAR o objecto	"Ligar o A/C com o switch externo e não abrir a unidade" esconde como flui a eletricidade da unidade ao motor
Herança	Objectos que pertençem a uma dada classe, herdan as propriedades dessa classe e das suas super-classes (classes que contém essa classe)	Dado que o HEP43X é um filtro de ar, herda todas as propriedades dos filtros de ar i.e. um filtro e um ventilador
Generalização	Identificar as propriedades comuns dum grupo de objectos	Todos os filtros de ar tem um filtro para limpar o ar e uma ventoinha para fazer circular o ar
Especialização	Identificar as propriedades diferentes de um objecto (ou grupo de objectos)	O HEP43X é único porque tem um sensor de movimento que acelera a ventoinha quando há muito pó no ar.
Agregação	Separar o todo das partes	O filtro de ar como um todo suga o ar para o devolver limpo. Agora o filtro tem 2 filtros, uma ventoinha, o motor da ventoinha, um switch e alguns cabos, cada um com uma função específica.
Polimorfismo	Significa "muitas formas". Um objecto com o mesmo comportamento pode realizá-lo de diversas formas	Todos os filtros contém a operação "ligar", mas podem ligar através de mecanismos diferentes.

UML faz parte da análise e desenho OO, a qual tem os seguintes objetivos:

1. Identificar as classes de objetos de um sistema
2. Identificar os atributos e operações dessas classes
3. Identificar as suas inter-relações e interações
4. Identificar os ciclos de vida dos objetos

5. Definir um desenho que organize as classes passível de ser implementado
6. Converter o desenho num programa executável usando linguagens orientadas a objetos

Java

Exceções

Uma exceção (ou evento excecional) é um problema que surge durante a execução de um programa. Quando ocorre uma exceção, o fluxo normal do programa é interrompido e o programa / aplicativo termina de forma anormal, o que não é recomendado; portanto, essas exceções devem ser tratadas.

Uma exceção pode ocorrer por vários motivos diferentes. A seguir, estão alguns cenários em que ocorre uma exceção.

- Um usuário inseriu dados inválidos.
- Um arquivo que precisa ser aberto não pode ser encontrado.
- Uma conexão de rede foi perdida no meio das comunicações ou a JVM ficou sem memória.

Algumas dessas exceções são causadas por erro do usuário, outras por erro do programador e outras por recursos físicos que falharam de alguma maneira.

Com base nisso, temos três categorias de exceções. Você precisa entendê-los para saber como o tratamento de exceções funciona em Java.

- Exceções verificadas - Uma exceção verificada é uma exceção que é verificada (notificada) pelo compilador no momento da compilação; elas também são chamadas de exceções no tempo de compilação. Essas exceções não podem ser simplesmente ignoradas, o programador deve cuidar (manipular) essas exceções.

Classe

Uma classe é um blueprint ou protótipo definido pelo usuário a partir do qual os objetos são criados. Representa o conjunto de propriedades ou métodos comuns a todos os objetos de um tipo. Em geral, as declarações de classe podem incluir esses componentes, em ordem:

Modificadores: uma classe pode ser pública ou ter acesso padrão (consulte isso para detalhes).

- Nome da turma: o nome deve começar com uma letra inicial (maiúscula por convenção).
- Superclasse (se houver): o nome do pai da classe (superclasse), se houver, precedido pela palavra-chave se estende. Uma classe só pode estender (subclasse) um pai.
- Interfaces (se houver): uma lista separada por vírgula de interfaces implementadas pela classe, se houver, precedida pela palavra-chave implementada. Uma classe pode implementar mais de uma interface.
- Corpo: o corpo da classe cercado por chaves {}.

Construtores são usados para inicializar novos objetos. Campos são variáveis que fornecem o estado da classe e seus objetos, e métodos são usados para implementar o comportamento da classe e de seus objetos. [2]

Objeto

É uma unidade básica de programação orientada a objetos e representa as entidades da vida real. Um programa Java típico cria muitos objetos que, como você sabe, interagem invocando métodos. Um objeto consiste em:

Estado: é representado pelos atributos de um objeto. Também reflete as propriedades de um objeto.

Comportamento: é representado pelos métodos de um objeto. Também reflete a resposta de um objeto com outros objetos.

Identidade: atribui um nome exclusivo a um objeto e permite que um objeto interaja com outros objetos.

Quando um objeto de uma classe é criado, é dito que a classe é instanciada. Todas as instâncias compartilham os atributos e o comportamento da classe. Mas os valores desses atributos, ou seja, os estados são únicos para cada objeto. Uma única classe pode ter qualquer número de instâncias.[3]

Resolução

Numa primeira fase foi feito um diagrama onde procedemos à organização das classes e como estas se iriam relacionar entre si e quais os métodos associados a cada uma.

Depois de feito o diagrama, procedemos à realização do código em si, onde, no geral, não houve problemas de implementação que requeressem uma atenção demasiado prolongada.

A primeira classe implementada foi a classe ponto, classe esta que foi reutilizada do lab1.

De seguida foi feita a classe circunferência onde estão implementados métodos para calcular o centro usando a técnica indicada no enunciado e algumas outras funcionalidades necessárias como getRaio e getCentro para podermos ter acesso a esses valores, apesar de estes não serem necessários.

Passando para a classe Segmento Reta onde foi implementado o método inversa que tem como função calcular a mediatriz, para tal foram também implementado outro método, esse método calcula o declive do segmento de reta no qual terá como objetivo para facilitar a descoberta do declive da reta mediatriz e a ordenada da mesma, por fim foi feito um método ponto medio para saber qual seria a equação da reta inversa. Aqui também foram implementados métodos de auxílio para devolver os pontos do segmento de reta.

Por fim, foi feita a classe reta onde iria ter o método interseção para chegar-mos ao centro da circunferência, este método deu alguns problemas porque causa das retas que seria paralelas ao eixo das ordenadas, para isso foi feito um construtor que receberia um argumento

que seria o valor do X e uma variável para verificar se a reta era paralela ao eixo das ordenadas, depois foi usado uma formula para encontrar o ponto de interseção. Nesta classe também foi feito métodos auxiliares com declive e ordenada que devolvem os seus valores (não foi feito para o X).

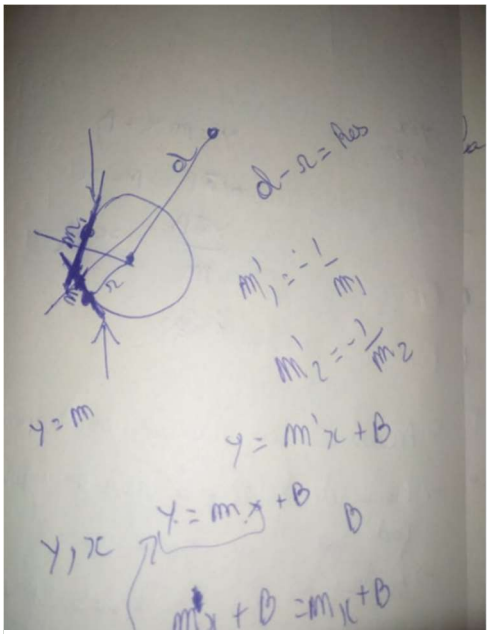


Figura 2-interseçao das retas

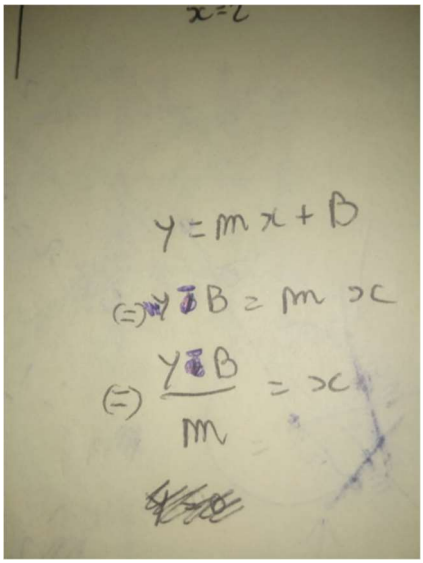
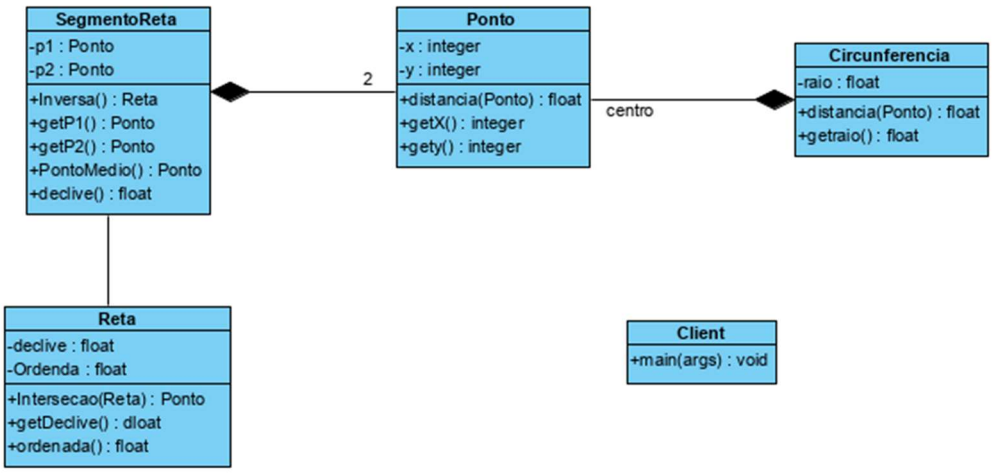


Figura 1-inversa de um segmento reta

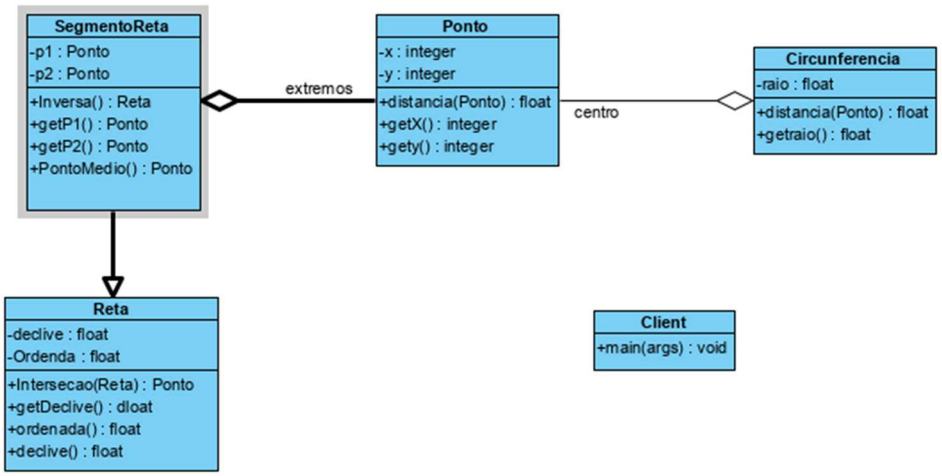
Desenvolvimento do UML

Após definidos os métodos e as variáveis de cada classe, começamos por um UML bastante rudimentar em que a classe "*SegmentoReta*" e a classe "*Circunferencia*" estavam relacionadas ambas à classe "Ponto" por composição e existia uma relação de associação entre "SegmentoReta" e "Reta", alterando só as ligações.

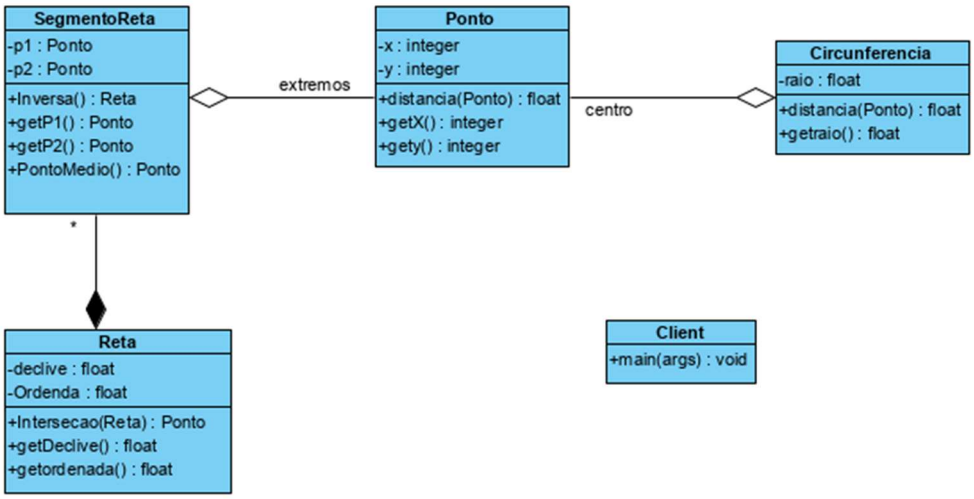
1º versão do UML



2º versão do UML



3º versão do UML

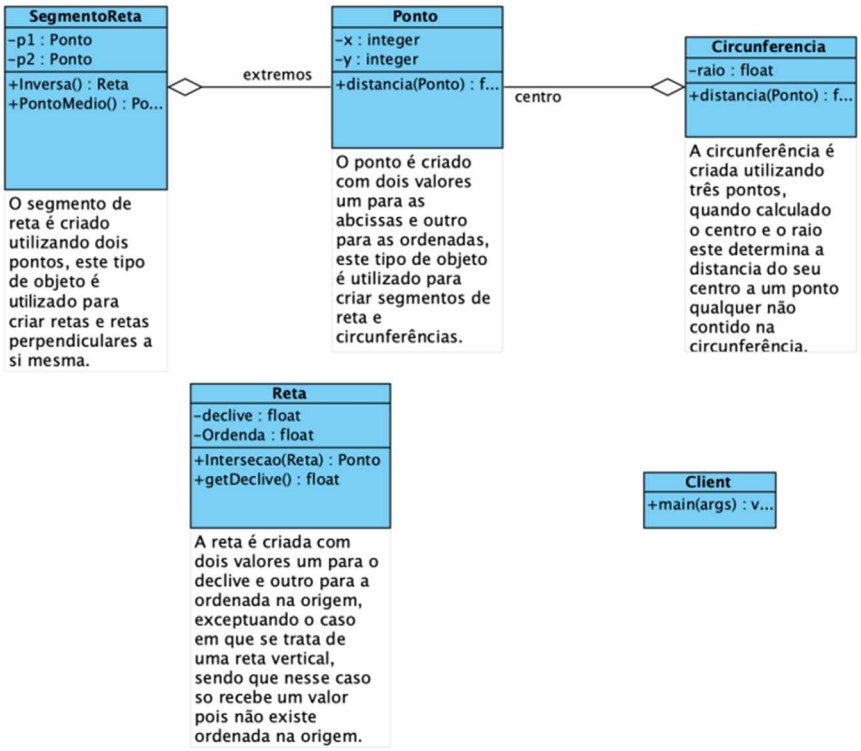


Após três versões de UML, chegamos ao nosso UML final, no qual os getters da Classe foram retirados, à exceção do getDeclive(), que decidimos manter, pois existiam cálculos a ser realizados.

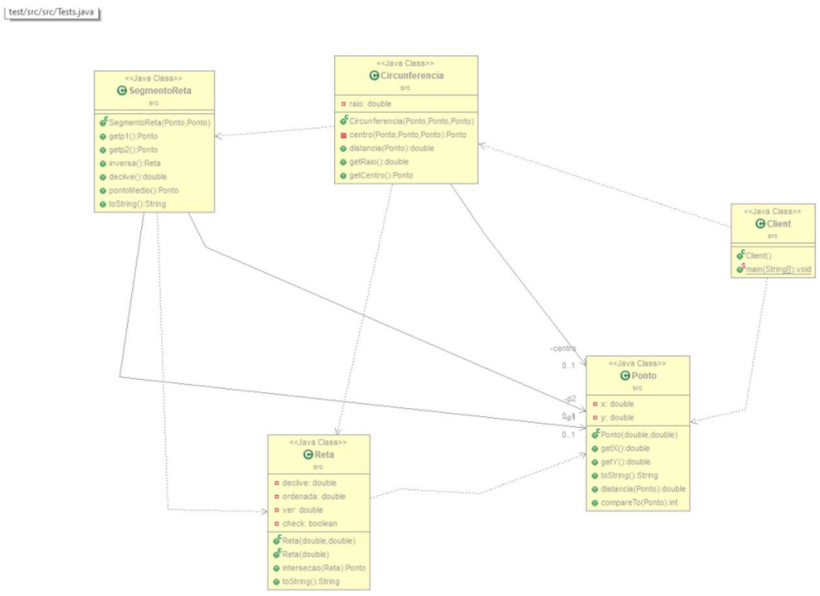
Relativamente às relações, optámos por retirar a relação da Classe "SegmentoReta" e "Reta" e alterámos as relações de "Ponto" para "*SegmentoReta*" e "*Circunferencia*" de relações de composição para relações de agregação, pois nomeadamente, se a Classe "*SegmentoReta*" ou "*Circunferencia*" desaparecerem a Classe Ponto não desaparece. Sugestao dada pelo professor JVO.

Em seguida foi adicionada uma descrição a cada uma das classes para a compreensão das mesmas

Versão Final do UML -1º Parte



UML de Implementação após o código estar escrito



O diagrama de implementação foi ao encontro do nosso diagrama inicial, nomeadamente ao encontro do diagrama Versão Final UML- 1ºParte

Casos de uso

Os testes unitários utilizados foram determinantes para verificar os resultados obtidos durante a execução do programa. O ficheiro que contêm estes testes chama-se "Teste.java" (Ver no Anexo). Numa primeira fase criámos testes unitários para verificar em que posição estava num determinado Segmento de Reta, Reta, Ponto e Circunferência. Nestes testes unitários utilizamos o método assertEquals() para comparar os resultados obtidos no programa com o verdadeiro resultado esperado.

Verificamos quando uma reta era vertical ou não através da função teste intersecaoTests0 usando o método intersecao(reta). Também foi feito para o teste testdistanciaAcircunerencia().

Conclusão

Após feito o código e gerado o respetivo UML, deparamo-nos que o nosso UML inicial foi bem deduzido, não havendo alterações significativas em relação ao gerado pela implementação. As alterações feitas, foram acrescentar uma variável check à classe "*Reta*" para verificar se era uma reta vertical e o método CompareTo ao ponto para comparação de dois pontos.

Em suma, acreditamos que a implementação em si está realizada de forma clara e eficiente não gerando confusão nem questões a quem se digne a fazer a sua verificação.

Bibliografia

1. http://w3.ualg.pt/~mzacaria/tutorial-uml/topico-intro.html
2. https://www.tutorialspoint.com/java/java_exceptions.htm
3. https://www.geeksforgeeks.org/classes-objects-java/

Anexo A

Descrição do problema

Dada uma circunferência c e um ponto P, com garantia de estar fora de c, determine a distância d, de P a c. A figura mostra uma instância do problema.

Se os dois pontos forem dados por suas coordenadas $(x1, y1)$ e $(x2, y2)$, respetivamente, a distância d é dada por: $d = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$

A circunferência é definida por três de seus pontos, indicados por A, B e C, respetivamente. Use o método descrito na figura abaixo para calcular o centro X de a circunferência. O centro X é dado pela intercessão dos dois mediadores r 'e s'. Lembre-se de que a inclinação m de uma linha definida pelos pontos $(x1, y1)$ e $(x2, y2)$ é dada por. As inclinações m' e m de duas linhas perpendiculares satisfazem m O método acima deve apoiar qualquer orientação das mediatrizes, incluindo linhas horizontais e verticais.

NB: Qualquer outro método para calcular a circunferência, independentemente do seu mérito, será citar com 0 (zero).

Input:

A entrada possui 4 linhas, cada uma delas contendo as coordenadas de um ponto. As 3 primeiras linhas referem-se aos três pontos da circunferência, enquanto o quarto é o ponto P, fora da circunferência.

As coordenadas de um ponto são números reais separados por espaço.

Output

A saída é um número real arredondado de duas casas decimais com o número necessário

distância ou uma mensagem de erro.

Sample Input 1

0.0 0.0

0.0 2.0

2.0 2.0

3.0 3.0

Sample Output 1 1.41

Sample Input 2

0.0 2.0

0.0 0.0

2.0 2.0

3.0 3.0

Sample Output 2 1.41

Sample Input 3

0.0 1.0

1.0 2.0

1.0 0.0

1.0 3.0

Sample Output 3 1.00

Sample Input 4

0.0 -1.0

-1.0 -2.0

-1.0 0.0

-1.0 -3.0

Sample Output 4 1.00

Sample Input 5

0.0 0.0

1.0 1.0

2.0 2.0

4.0 -5.0

Sample Output 5 invalid points

Sample Input 6

2.0 3.0

2.0 3.0

-2.0 -2.0

2.4 1.3

Sample Output 6 invalid points