

Programação Orientada pelos Objectos

1º Teste (12/Abril/2013)

LEI 2012/2013

Instruções:

- Antes de começar a resolver, **leia o enunciado do princípio até ao fim.**
 - As interfaces e classes do grupo de I e II têm mais métodos do que os que deverá implementar na resolução deste teste. Em cada grupo, tenha o cuidado de **ver com muita atenção quais os métodos que deve implementar**, para **não desperdiçar o seu tempo a implementar métodos que não lhe são pedidos.**
 - Disponibilizamos a descrição sumária de todos os métodos, incluindo os que não tem de implementar, para que os possa usar na sua resolução.
 - **Pode** usar caneta ou lápis.
 - Não é permitido consultar quaisquer elementos para além deste enunciado.
-

No grupo I e II terá que implementar parcialmente algumas das classes necessárias à construção de um programa para manter uma colecção de formas geométricas. Note que apenas é permitido acrescentar métodos privados às classes (não pode alterar/acrescentar variáveis ou métodos não privados).

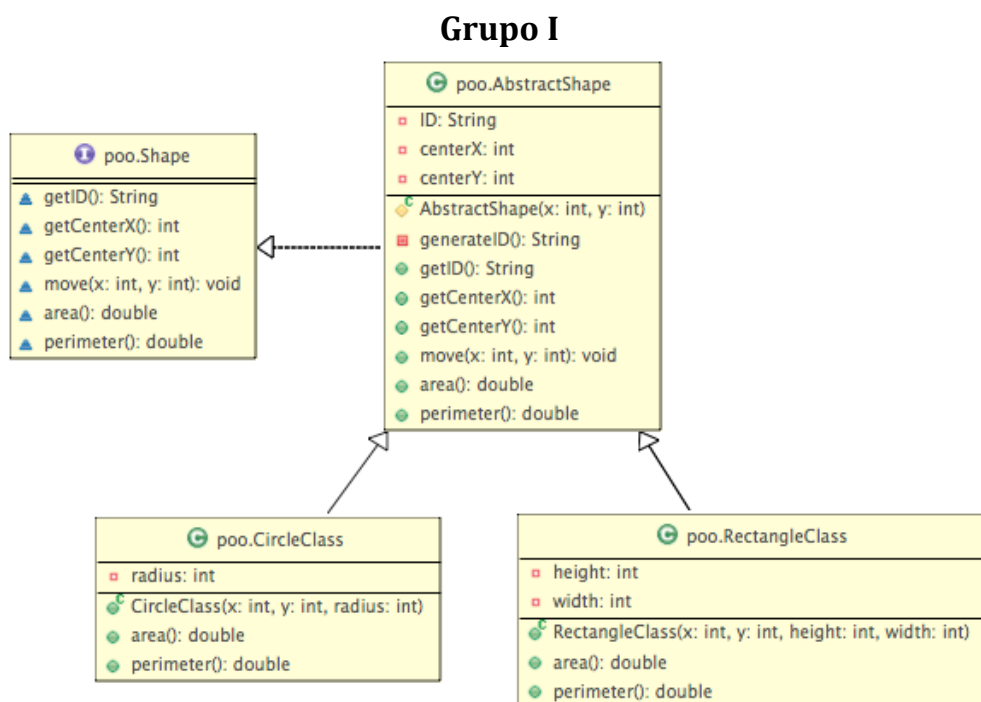


Figura 1 – Formas geométricas.

A interface `Shape` representa uma forma geométrica num plano cartesiano (Fig. 1). A interface `Shape` tem os seguintes métodos:

- `getID` devolve o identificador da forma geométrica;
- `getCenterX` devolve a abcissa do centro da forma geométrica;
- `getCenterY` devolve a ordenada do centro da forma geométrica;
- `move` altera o centro da forma geométrica para os valores recebidos como argumento;
- `area` devolve a área da forma geométrica;
- `perimeter` devolve o perímetro da forma geométrica.

A classe abstracta `AbstractShape` implementa a interface `Shape`. Nesta classe temos um construtor `AbstractShape` que recebe o valor da abcissa e da ordenada do centro. O identificador único de cada forma geométrica (variável `ID`) é gerado pelo método privado `generateID`. Os métodos `getID`, `getCenterX`, `getCenterY`, `move`, `area` e `perimeter` têm o comportamento já descrito na explicação dada sobre a interface `Shape`.

Implemente os seguintes métodos das classes `AbstractShape`, `RectangleClass` e `CircleClass`.

- a) O construtor `AbstractShape(int x, int y)`.
- b) O construtor `CircleClass (int x, int y, int radius)`.
- c) O método `void move(int x, int y)` da classe `AbstractShape`.
- d) O método `double perimeter()` da classe `RectangleClass`.

Grupo II

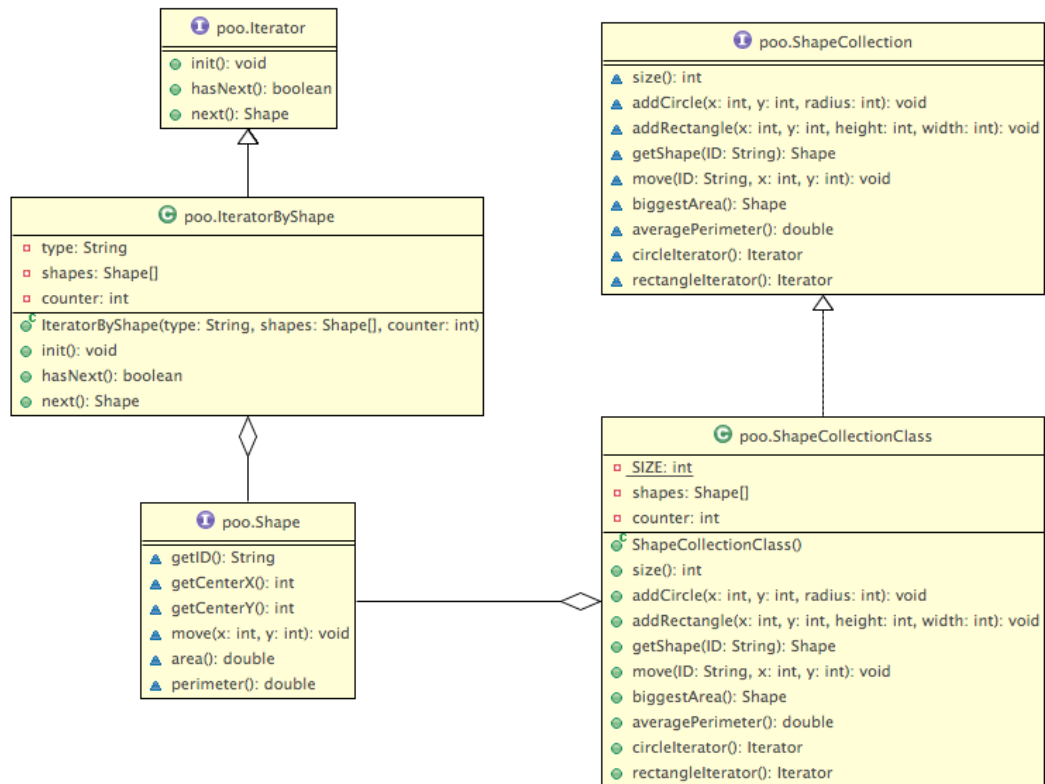


Figura 2 – Colecção de formas geométricas.

A interface `ShapeCollection` representa uma colecção de formas geométricas (Fig. 2):

- `addCircle` recebe a abcissa e a ordenada do centro, seguido do raio e adiciona um novo círculo à colecção. Se o vector estiver cheio, o método não faz nada. Considere que podem existir formas geométricas iguais na colecção;
- `addRectangle` recebe a abcissa e a ordenada do centro, seguido da altura e a largura e adiciona um novo rectângulo à colecção. Se o vector estiver cheio, o método não faz nada. Considere que podem existir formas geométricas iguais na colecção;
- `getShape` procura na colecção a forma geométrica com o ID dado. Caso o ID não exista, o método devolve `null`;
- `move` altera o centro da forma geométrica com o ID dado. Caso o ID não exista, o método não faz nada;
- `biggestArea` devolve a forma geométrica da colecção com a maior área. Se existirem várias formas geométricas que satisfaçam este critério, o método deve devolver a forma geométrica mais recente. Considere que este método só é invocado quando a colecção não é vazia.
- `averagePerimeter` determina a média dos perímetros das formas geométricas da colecção. Considere que este método só é invocado quando a colecção não é vazia.
- `circleIterator` devolve um iterador que permite iterar sobre os círculos da colecção;
- `rectangleIterator` devolve um iterador que permite iterar sobre os rectângulos da colecção.

Implemente os seguintes métodos da classe `ShapeCollectionClass`:

- O método `void addRectangle(int x, int y, int height, int width)`.
- O método `void move(String ID, int x, int y)`.
- O método `Shape biggestArea()`.
- O método `Iterator rectangleIterator()`.

Grupo III

Pretende-se desenvolver um programa para a gestão de uma lista de tarefas. Esta aplicação deve permitir:

- adicionar uma tarefa de uma determinada categoria. Considere que existem as categorias emprego, família, amigos;
- completar tarefas;
- listar tarefas por categoria (e.g. listar todas as tarefas relacionadas com o emprego);
- listar tarefas prioritárias. Considere que todas as tarefas da categoria emprego com prioridade 1 e 2 devem ser listadas. Sendo que para as restantes categorias, apenas tarefas de prioridade 1 são consideradas;
- listar tarefas já completadas;

Considere que cada tarefa tem uma descrição (representada por uma String), a data limite (representada por uma String), a prioridade (valor de 1 a 5, sendo o valor 1 o mais prioritário) e um valor booleano que indica se já foi ou não concluída. As tarefas são univocamente identificadas pela descrição e pela data limite. Considere que não podem haver tarefas repetidas na lista.

Apresente a sua proposta de modelação para o programa, através de um **diagrama de classes e interfaces**, tendo em atenção que deve incluir na sua resposta:

- a interface de topo – *ToDoList* – com a qual o programa principal irá interagir. Esta interface deve ser completamente especificada.
- as variáveis de instância da classe que implementa a interface *ToDoList*.
- para as restantes componentes do diagrama, isto é, para as interfaces e classes que não a interface *ToDoList*, omita a indicação das operações e das variáveis de instância.

Nota 1: Não é necessário implementar nenhuma das operações.

Nota 2: Apresente de forma distinta classes e interfaces, identifique caso existam classes abstractas e indique a visibilidade das variáveis de instância.

Nota 3: Para efeitos de clareza da apresentação represente as relações com uma seta etiquetada, por exemplo $A \xrightarrow{\text{extends}} B$ significa A extends B (analogamente para implements, eventualmente com linha tracejada mas sempre etiquetada com implements).

