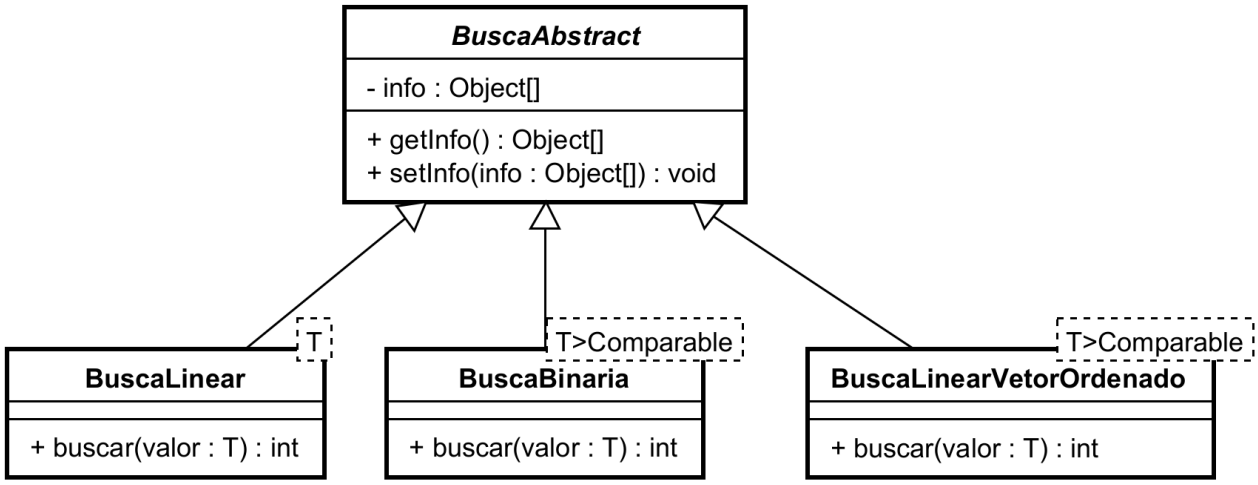
Lista de Exercício 11

Questão 1

Construir o projeto de software abaixo, a fim de implementar os algoritmos de busca vistos em sala de aula.



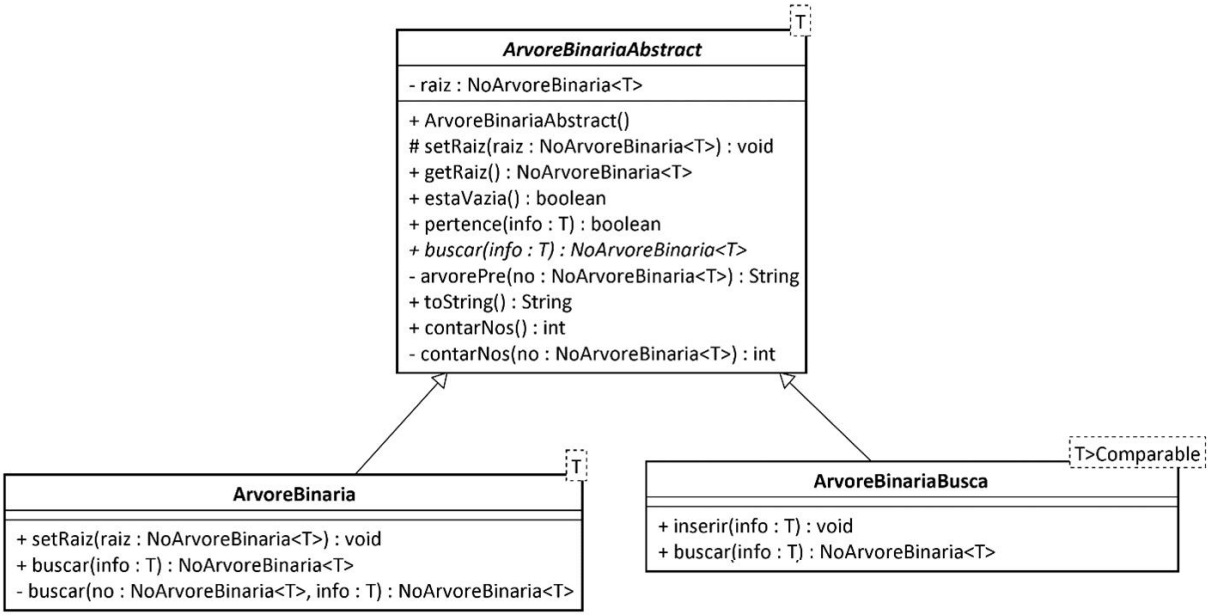
Questão 2

Implementar o seguinte plano de testes:

Plano de testes PL01 – Validar funcionamento dos algoritmos			
Caso	Descrição	Entrada	Saída esperada
1	Conferir que o método buscar() localiza um dado armazenado através do algoritmo de busca linear.	Criar lista ordenada de números inteiros e adicionar os seguintes dados, nesta ordem: 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100.	O método buscar(20) deve resultar em 2.
2	Conferir que o método buscar() localiza um dado armazenado através do algoritmo de busca linear em vetor ordenado.	A entrada é idêntica ao caso 1.	O método buscar(40) deve resultar em 4.
3	Conferir que o método buscar() localiza um dado armazenado através do algoritmo de busca binária	A entrada é idêntica ao caso 1	O método buscar(70) deve resultar em 7.
4	Conferir que o método buscar() localiza um dado armazenado através do algoritmo de busca binária	A entrada é idêntica ao caso 1	O método buscar(75) deve resultar em -1.

Questão 3

O objetivo desta atividade prática é realizar a implementação de árvores binárias de busca, de acordo com o diagrama de classes da figura abaixo.



Para construir a solução, vamos aproveitar a implementação do exercício 8. Para isso, vamos modificar nossa implementação de árvore binária para que seja extensível. Veja as etapas:

a) Copiar as classes **`ArvoreBinaria`** e **`NoArvoreBinaria`** do exercício 8 para um novo projeto;
b) Renomear a classe **`ArvoreBinaria`** para **`ArvoreBinariaAbstract`**;
c) Tornar a nova classe **`ArvoreBinariaAbstract`** uma classe abstrata;
d) Criar o método abstrato **`buscar()`** na classe **`ArvoreBinariaAbstract`**;
e) Implementar o método **`pertence()`** na classe **`ArvoreBinariaAbstract`** para reusar o método **`buscar()`**;
f) Tornar o método **`setRaiz()`**, da classe **`ArvoreBinariaAbstract`**, protegido;
g) Criar a classe **`ArvoreBinaria`** estendendo-a da classe **`ArvoreBinariaAbstract`**;
h) Inserir o método **`setRaiz()`** na classe **`ArvoreBinaria`** e implementar seu código para reutilizar a implementação do método **`setRaiz()`** da superclasse.
i) Implementar o método **`buscar()`** na classe **`ArvoreBinaria`**, para que localize se há um nó que armazene o dado fornecido como argumento. Em caso positivo, o método **`buscar()`** deve retornar este nó, caso contrário, deverá retornar **`null`**.

Os métodos novos para serem implementados na classe **`ArvoreBinariaBusca`** são:

a) **`inserir()`**: este método deve inserir o dado, fornecido como argumento, na árvore binária de busca. O método deve armazenar o dado num nó de forma que a árvore binária mantenha as características de uma "arvore binária de busca".
b) **`buscar()`**: este método deve buscar o dado fornecido como argumento, na árvore binária, retornando o nó que o armazena. Utilizar o algoritmo de busca binária em árvore.

Questão 4

Implemente o seguinte plano de testes.

Plano de testes PL01 – Validar funcionamento da implementação de árvore binária de busca			
Caso	Descrição	Entrada	Saída esperada
1	Conferir se o método inserir() mantém os dados armazenados adequadamente, mantendo a	Criar uma árvore binária de inteiros e adicionar os seguintes dados, nesta ordem: 50,30,70,40,25,75,65,35,60	O método toString() deve retornar: <50<30<25<><>><40<35<><>><>>><70<65<60<><><>><>><75<><>>>>

	árvore com a característica de ser uma árvore binária de busca.		
2	Conferir se a árvore consegue remover um nó folha	Criar uma árvore inserindo os dados nesta ordem: 50,30,25,40 Remover o nó 40	O método toString() deve retornar: <50<30<25<><><><><>
3	Conferir se a árvore consegue remover nó com um filho	Criar uma árvore inserindo os dados nesta ordem: 80,52,90,48,71,63,67 Remover o nó 71	O método toString() deve retornar: <80<52<48<><><><63<><67<><><><><90<><><>