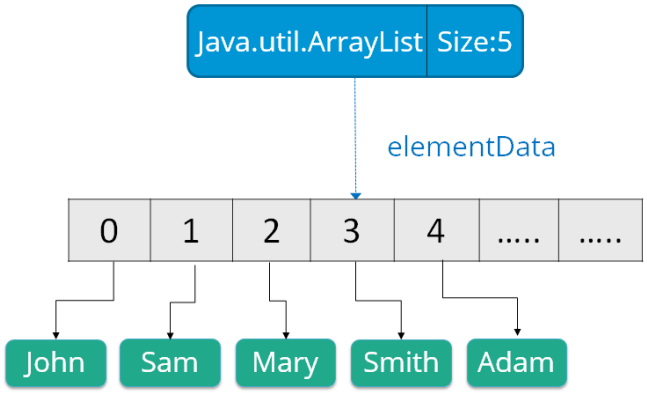
Coleção de Elementos[1]

Em uma linguagem orientado a objetos é comum trabalharmos não apenas com um objeto, mas com um conjunto de objetos. Uma ***collection*** (coleção) é um objeto que agrupa múltiplos objetos ou tipos primitivos dentro de uma única unidade.



Na linguagem Java para armazenar uma coleção de objetos temos a **classe ArrayList** (implementada como vetor), cada objeto na lista é acessado através de um índice e são chamados **elementos** ou **itens**. A classe ArrayList também disponibiliza uma série de métodos para inserção, remoção e consulta dos elementos da coleção, por exemplo:

- **boolean add(Object elemento)**: método adiciona um elemento no final da coleção.
- **void add(int index, Object elemento)**: método insere um elemento antes do índice informado pelo parâmetro index.
- **int indexOf(Object element)**: método retorna a posição da primeira ocorrência do elemento contido na coleção.
- **Object remove(int index)**: Remove o elemento no índice informado pelo parâmetro index.

Objetivo do trabalho

O objetivo desta atividade é implementar um programa (=cliente) para administrar coleção elementos em uma linguagem orientada a objetos. Os elementos da coleção serão objetos da classe Ponto conforme descrito abaixo, **não é permitida** a utilização de classes já prontas da linguagem Java, tais como: **`Point`, `Point2D`** ou **`Point3D`**.

```
public class Ponto{
   private int x,y;
   public Ponto(int x, int y) {
        this.x = x;
        this.y = y;
   }
   . . . . . .
}
```

[1] **Importante**: A especificação desse trabalho pode sofrer modificações de acordo com discussões que tivermos em sala de aula.

Para armazenar a coleção de pontos **não é permitido** utilizar classes prontas da linguagem Java, como **ArrayList**, assim implementaremos a classe **`listaPonto`**, de tal forma que para cada instância da classe **`listaPonto`** teremos **N** instâncias da classe **`Ponto`**, ou seja, um relacionamentos com multiplicidade 1:N entre a classe **`listaPonto`**(1):**`Ponto`**(N). Para tanto, na classe **`listaPonto`** alocaremos um *container* (=vetor) de objetos da classe **`Ponto`** com **N** posições, mas nem sempre teremos **N** objetos no vetor, assim classe **`listaPontos`** deverá controlar a quantidade de objetos armazenados (**válidos**) no *container*.

```
public class listaPonto{
    private Ponto pontos[];
    private int validos;
    public ListaPonto( int N ){
        thils.pontos =new Ponto[N];
        this.validos = 0;
    }
   . . . . . .
}
```

Os elementos no *container* são armazenados de forma **continua e estável**, ou seja, não pode existir **"buracos"** no meio do *container* e as operações para adicionar e de remover um elemento não **alteram a posição relativa** dos elementos no *container* e, caso tenham sucesso deve ser atualizada a quantidade de elementos válidos na coleção. No programa cliente o usuário pode solicitar as seguintes operações.

1. Adicionar um elemento no final da coleção;
2. Adicionar um elemento em uma posição da coleção;
3. Retornar o índice da primeira ocorrência de um elemento especificado na coleção.
4. Remover um elemento em uma posição na coleção.
5. Calcular a distância dos dois pontos mais distantes na coleção;
6. Retornar uma coleção de pontos contido em uma circunferência.

Para o usuário escolher uma das operações, o seu programa deverá mostrar um **menu de opções**, sendo que para cada opção no menu deverá ser **executada a operação correspondente**, por exemplo, para adicionar um elemento no final da coleção, o usuário deve escolher a opção **1-Adicionar no final** e digitar um elemento que em seguida será informado para método implementado na classe **`listaPonto`** que implementa a operação correspondente. O programa **deverá executar continuamente** até o momento que usuário solicite finalizar a aplicação, ou seja, o usuário pode escolher outras operações do **menu de opção** várias vezes. Ao final da execução de cada operação **é impresso** os elementos **válidos na coleção**. A seguir são descritas detalhadamente as operações, para **cada operação** deverá ser implementado **um método** correspondente na classe **`listaPonto`** e **não é permitido o uso de variáveis globais**:

1. Adicionar um elemento no final da coleção.

Adiciona um novo elemento na primeira posição disponível no final do vetor. Caso o *container* já esteja no limite, ou seja, com **N** elementos, o elemento não é adicionado e a quantidade de elementos válidos não é modificada, caso seja inserido o elemento a quantidade de elementos válidos é incrementada. O método responsável em implementar essa operação receberá como parâmetro o novo elemento, ou seja, um objeto da classe **`Ponto`**.

2. **Adicionar um elemento em uma posição da coleção.**

   A operação de adicionar um elemento em uma determinada posição é mais delicada. Primeiro, precisamos verificar se a posição faz sentido ou não, ou seja, só podemos adicionar um elemento em alguma posição que já estava ocupada ou na primeira posição disponível no final do *container*. Caso a posição seja válida, devemos tomar cuidado para não colocar um elemento sobre outro. É preciso deslocar todos os elementos a "direita" da posição onde vamos inserir o elemento uma vez para a "frente". Isso abrirá um espaço para guardar o novo elemento. Se conseguirmos inserir a elemento a quantidade de elementos válidos no containre é atualizada. O método responsável em implementar essa operação receberá como parâmetro o novo elemento, ou seja, um objeto da classe **`Ponto`** e o índice onde será inserido o elemento.

3. **Retornar o índice da primeira ocorrência de um elemento especificado na coleção.**

   Nessa operação será informado para método o elemento (Ponto) que será buscado no vetor, e em seguida, o método o índice onde o elemento foi encontrado. Caso o elemento não esteja no vetor o método devolve -1.

4. **Remover um elemento em uma posição na coleção;**

   Para essa operação precisamos verificar se a posição está ocupada ou não, se a posição estiver ocupada, então podemos remover o elemento. Para isso basta deslocar os elementos que estavam à direita daquele que removemos uma vez para esquerda e fechamos o "buraco" aberto pela remoção e a quantidade de elementos no vetor é atualizada. O método responsável em implementar essa operação receberá como parâmetro o índice do elemento que será removido.

5. **Calcular a distância dos dois pontos mais distantes na coleção;**

   O método que implementa essa operação deverá retornar o valor da distância entre os dois pontos mais distantes armazenados na coleção, não sendo necessário passar nenhum parâmetro para o método.

6. **Retornar uma coleção de pontos contido em uma circunferência.**

   E por fim, temos a operação que encontra pontos que estão contidos dentro de uma circunferência, o método receberá por parâmetro o valor do raio (double) e o centro da circunfência (Ponto), e devolverá uma nova coleção de pontos, ou seja, um objeto da classe **`listaPonto`** contendo os pontos cuja distância do centro da circunferência é menor ou igual ao comprimento do raio. No programa cliente é impresso os pontos retornados.

Restrições do projeto

1. O programa deve estar bem documentado e implementado na **linguagem Java**, aplicando os conhecimentos sobre orientado a objetos vistos na disciplina, tais como: atributos encapsulados (privados), passagem de parâmetro implícita e explicita.
2. Este trabalho pode ser desenvolvido em grupos de até 2 alunos e sigam as **Orientações para Desenvolvimento de Trabalhos Práticos** disponível no **Blackboard**.
3. A entrega do trabalho deve ser feita pelo **Blackboard** (não serão aceitos trabalhos entregues via e-mail) e será avaliado de acordo com os seguintes critérios:
   - Funcionamento do programa;
   - O quão fiel é o programa quanto à descrição do enunciado;
   - Indentação, comentários e legibilidade do código;
   - Clareza na nomenclatura de variáveis;