ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO
Departamento de Engenharia de Computação e Sistemas Digitais

PCS 3111 - LABORATÓRIO DE PROGRAMAÇÃO ORIENTADA A OBJETOS PARA A ENGENHARIA ELÉTRICA

EXERCÍCIO PROGRAMA 2 – 2017

Resumo

O objetivo do EP2 é exercitar os conceitos de Orientação a Objetos ao melhorar o *Visualizador Científico* implementado no EP1 para permitir a criação de séries envolvendo tempo e a geração de gráficos.

1 Introdução

No EP1 foram implementadas as classes **Serie** e **Ponto**, permitindo visualizar *textualmente* os dados obtidos por um microcontrolador. No EP2 essas classes serão melhoradas e outras classes serão adicionadas para permitir duas novas funcionalidades. A primeira é permitir que uma **Serie** tenha o tempo como coordenada horizontal. A segunda funcionalidade é permitir visualizar *graficamente* uma **Serie** obtida pelo microcontrolador. Os gráficos serão gerados usando um *framework* para desenvolvimento de aplicações em C++ chamado Qt (https://www.qt.io), o qual pode ser usado gratuitamente para o desenvolvimento de projetos *open source*.

A solução deve empregar adequadamente conceitos de Orientação a Objetos apresentados na disciplina: classe, objeto, atributo, método, encapsulamento, herança e polimorfismo – o que representa o conteúdo até a Aula 7. A qualidade do código também será avaliada.

2 Projeto

Deve-se implementar em C++ as classes **Grafico**, **Eixo**, **EixoDinamico** e **SerieTemporal**, além das classes **Serie** e **Ponto** que serão melhoradas considerando os novos conceitos apresentados na disciplina e para permitir as novas funcionalidades. Também será necessário melhorar o `main` de forma a criar o programa da forma desejada.

Atenção: o nome das classes e a assinatura dos métodos **devem seguir exatamente** o especificado neste documento. As classes não devem possuir outros membros (atributos ou métodos) **públicos** além dos especificados, a menos dos métodos definidos na classe pai e que precisaram ser redefinidos. Note que você poderá definir atributos e método protegidos e privados, conforme necessário.

Também não é permitida a criação de outras classes além dessas.

Cada uma das classes deve ter um arquivo de definição (".h") e um arquivo de implementação (".cpp"). Os arquivos devem ter exatamente o nome da classe. Por exemplo, deve-se ter os arquivos "EixoDinamico.cpp" e "EixoDinamico.h". Porém, diferentemente do EP1 não será entregue um código junto com o enunciado; você deve criar os arquivos necessários.

Note que as classes definidas no EP1 foram melhoradas, adicionando construtores e removendo métodos *getters* e *setters* desnecessários considerando o conceito de encapsulamento. Outras alterações também foram necessárias para permitir as funcionalidades especificadas. Portanto será necessário corrigir as classes já feitas.

A interface gráfica será gerada pela classe **Tela** disponibilizada e explicada na Seção 4. Não use outros recursos disponíveis pelo *framework* Qt.

2.1 Classe Ponto

A classe **Ponto** desenvolvida no EP1 foi melhorada ao adicionar um construtor recebendo as coordenadas e removendo os métodos *setters* – que não são úteis para esta aplicação. Com isso, os únicos métodos públicos que a classe deve possuir são:

```
/**
 * Cria um Ponto informando as coordenadas x e y.
 */
Ponto(double x, double y);
virtual ~Ponto();

/**
 * Obtém o valor do Ponto na coordenada horizontal (x).
 */
double getX();

/**
 * Obtém o valor do Ponto na coordenada vertical (y).
 */
double getY();

/**
 * Imprime na saída padrão (cout) o Ponto no formato (x, y).
 * Pule uma linha após imprimir o Ponto.
 */
void imprimir();

/**
 * Informa se este Ponto é igual a outro.
 * Um Ponto é igual se os valores x e y dos Pontos são
 * suficientemente próximos.
 * @param outro Um outro Ponto.
 */
bool eIgual(Ponto* outro);
```

Os métodos `imprimir` e `eIgual` devem seguir a especificação do EP1.

2.2 Classe Serie

Assim como a classe **Ponto**, a classe **Serie** foi melhorada ao definir um construtor e remover os métodos *setters* que não eram necessários. Os demais métodos foram definidos como virtual e devem seguir a especificação definida no EP1. Com isso, os únicos métodos públicos que a classe deve possuir são:

```
/**
 * Cria uma Serie informando o nome dela e o nome dos canais X e Y.
 */
Serie(string nome, string nomeDoCanalX, string nomeDoCanalY);
virtual ~Serie();

// Permite obter o nome, o nomeDoCanalX e o nomeDoCanalY.
virtual string getNome();
virtual string getNomeDoCanalX();
virtual string getNomeDoCanalY();

/**
 * Informa a quantidade de pontos que a Serie possui.
 */
virtual int getQuantidade();
```

```
/**
 * Informa se a Serie está vazia.
 */
virtual bool estaVazia();

/**
 * Adiciona um novo ponto à Serie, informando sua coordenada x e y.
 *
 * Ignora o valor passado caso o NUMERO_MAXIMO_VALORES tenha
 * sido ultrapassado.
 */
virtual void adicionar(double x, double y);

/**
 * Obtém um ponto representando o limite superior da Serie.
 * A coordenada x desse ponto deve ser o máximo valor horizontal
 * existente na Serie e a coordenada y deve ser o máximo valor
 * vertical existente na Serie.
 *
 * Caso a Serie não tenha valores, deve-se retornar NULL.
 *
 * Por exemplo, para a Serie {(2, 3), (5, 1), (1, 2)} o limite
 * superior é (5, 3).
 */
virtual Ponto* getLimiteSuperior();

/**
 * Obtém um ponto representando o limite inferior da Serie.
 * A coordenada x desse ponto deve ser o mínimo valor horizontal
 * existente na Serie e a coordenada y deve ser o mínimo valor
 * vertical existente na Serie.
 *
 * Caso a Serie não tenha valores, deve-se retornar NULL.
 *
 * Por exemplo, para a Serie {(2, 3), (5, 1), (1, 2)} o limite
 * inferior é (1, 1).
 */
virtual Ponto* getLimiteInferior();

/**
 * Obtém o ponto que está na posição definida da Serie. A contagem de
 * posições começa em 0.
 *
 * Em caso de posições inválidas, retorne NULL.
 *
 * Por exemplo, para a Serie {(2, 3), (5, 1), (1, 2)}, getPosicao(0)
 * deve retornar (2, 3) e getPosicao(2) deve retornar (1, 2).
 */
virtual Ponto* getPosicao(int posicao);

/**
 * Imprime na saída padrão (cout) o nome da Serie e seus pontos
 * seguindo o formato definido.
 */
virtual void imprimir();
```

2.3 Classe SerieTemporal

A classe **SerieTemporal** representa uma **Serie** em que o tempo é a coordenada horizonal (por simplicidade o tempo não poderá ser a coordenada vertical). Isso permite que a passagem do tempo seja controlada pelo software, ao invés de ser controlada pela placa. Para implementar isso, a classe **SerieTemporal** deve ser subclasse da **Serie**. A seguir são apresentados os métodos públicos específicos a essa classe (note que a classe pode ter que redefinir métodos da classe pai):

```
/**
 * Cria uma SerieTemporal informando o nome da Serie e o nome do
 * canalY. O nome do canal X deve ser obrigatoriamente "Tempo".
 */
SerieTemporal(string nome, string nomeDoCanalY);
virtual ~SerieTemporal();

/**
 * Adiciona um novo Ponto a Serie, no instante seguinte ao do
 * ponto anterior adicionado por este método. O primeiro ponto
 * deve ser adicionado no instante 1.
 */
virtual void adicionar(double valor);

/**
 * Adiciona um Ponto a Serie informando a coordenada x e y.
 * Caso já exista um ponto cuja coordenada x seja suficientemente
 * próxima ao do valor x informado, ao invés de adicionar o ponto,
 * altere o valor anterior (mantendo sua posição no arranjo)
 * ao considerar a nova coordenada y.
 * Caso a coordenada x do ponto for < 1, o ponto não deve ser adicionado.
 */
virtual void adicionar(double x, double y);
```

Diferentemente de uma **Serie**, o construtor da **SerieTemporal** não deve receber o nome do canal X. Ele obrigatoriamente deve ser "Tempo".

O tempo deve ser contado a partir do 1. O método `adicionar(double valor)` deve adicionar um novo **Ponto** à **Serie**, no instante seguinte ao ponto anterior adicionado por este método (tempo + 1). O método `adicionar(double x, double y)` deve adicionar um novo **Ponto** com as coordenadas informadas. Caso o valor de x seja < 1, o **Ponto** não deve ser adicionado (não apresente um erro – apenas não adicione). Assim como no EP1, os valores devem ser adicionados até `NUMERO_MAXIMO_VALORES`. Caso se tente adicionar mais valores, eles devem ser ignorados

Uma restrição importante para uma **SerieTemporal** é que não devem existir dois **Pontos** em um mesmo instante de tempo. Para evitar esse problema, os métodos `adicionar` devem verificar, antes de adicionar, se já existe um **Ponto** cuja coordenada x é *suficientemente* próxima ao do valor de x a ser adicionado (ou o passado como parâmetro, no caso do método com dois parâmetros, ou o tempo seguinte ao do **Ponto** anteriormente adicionado pelo método com um parâmetro, no caso do método com um parâmetro). Caso exista, ao invés de adicionar um *novo* **Ponto**, deve-se alterar o valor da coordenada y desse **Ponto**, mantendo a sua posição no arranjo.

Como exemplo de funcionamento, considere uma **SerieTemporal** vazia. Ao chamar `adicionar(3)`, a **SerieTemporal** deve ficar com `{(1, 3)}`. Ao chamar `adicionar(4)`, a **SerieTemporal** deve ficar com `{(1, 3), (2, 4)}`. E ao chamar `adicionar(1, 5)`, a **SerieTemporal** deve ficar com `{(1, 5), (2, 4)}`, uma vez que já existe um valor com coordenada x = 1 na série. Ao chamar `adicionar(3, 9)`, a **SerieTemporal** deve ficar com `{(1, 5), (2, 4), (3, 9)}` e ao chamar `adicionar(6)`, a **SerieTemporal** deve ficar com `{(1, 5), (2, 4), (3, 6)}`. Assim como no EP1, considere que a coordenada é suficientemente próxima se a diferença entre os valores é menor que um ϵ (`1e-5`, por exemplo).

2.4 Classe Grafico

A classe **Grafico** é a responsável por desenhar um gráfico na **Tela** (a classe **Tela** é explicada na Seção 4). Para isso um **Grafico** possui dois **Eixos** (um horizontal e outro vertical) e uma **Serie**, que será desenhada. Os únicos métodos públicos que a classe deve possuir são:

```
/**
 * Cria um Grafico informando os Eixos e a Serie.
 */
Grafico(Eixo* x, Eixo* y, Serie* serie);
virtual ~Grafico();

Eixo* getEixoX();
Eixo* getEixoY();
Serie* getSerie();

/**
 * Desenha o Grafico na Tela.
 */
void desenhar();
```

Os métodos `getEixoX`, `getEixoY` e `getSerie` devem apenas retornar os valores definidos no construtor. O método `desenhar` deve criar uma **Tela**, definindo seus eixos, e plotar todos os pontos da **Serie**. A **Tela** deve ser então mostrada e depois destruída. Ou seja, esse método deve ser seguir o seguinte estilo:

```
Tela* t = ...
...
t->mostrar();
delete t;
```

Mais detalhes sobre o funcionamento da classe **Tela** são apresentados na Seção 4.

2.5 Classe Eixo

Um gráfico deve possuir dois **Eixos:** um das abscissas (horizontal, o "x") e um das ordenadas (vertical, o "y"). Cada **Eixo** possui um título e um valor mínimo e um valor máximo da escala, os quais não são mudados depois do objeto ser criado. Por exemplo, um **Eixo** horizontal pode apresentar valores de 0 a 5 e um **Eixo** vertical os valores de -5 a 5.

A seguir são apresentados os métodos públicos dessa classe:

```
/**
 * Cria um Eixo informando o título, o mínimo e o máximo.
 */
Eixo(string titulo, double minimo, double maximo);
virtual ~Eixo();

virtual string getTitulo();
virtual double getMinimo();
virtual double getMaximo();
```

Os método `getTitulo`, `getMinimo` e `getMaximo` devem apenas retornar os valores definidos no construtor.

2.6 Classe EixoDinamico

Um outro tipo de **Eixo** é o **EixoDinamico** (essa classe deve ser subclasse da classe **Eixo**). No **EixoDinamico**, o mínimo e o máximo do **Eixo** devem ser calculados a partir dos valores de uma **Serie**, a qual deve ser informada no construtor. O título do **EixoDinamico** deve ser o nome do canal

correspondente dessa **Serie** (se a orientação for horizontal, deve ser o nome do canal X; se for vertical, deve ser o nome do canal Y). A seguir são apresentados os métodos públicos específicos a essa classe[1] (note que a classe pode ter que redefinir métodos da classe pai):

```
/**
 * Cria um EixoDinamico informando o mínimo e o máximo padrão,
 * a Serie que será usada como base e a orientação (true se for
 * horizontal e false se for vertical).
 */
EixoDinamico(double minimoPadrao, double maximoPadrao,
             Serie* base, bool orientacaoHorizontal);
```

Caso a **Serie** possua *menos de 2 valores* ou *os valores mínimo e máximo sejam suficientemente iguais*, o **EixoDinamico** deve retornar os valores padrão, os quais são informados no construtor.

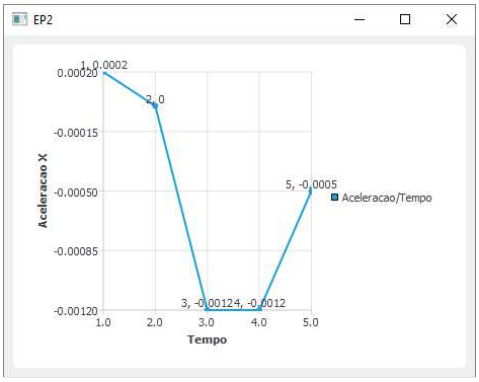
3 Interface com o microcontrolador

A classe **InterfaceSerial** foi melhorada ao definir um construtor e um destrutor. No construtor deve ser informada a porta a ser usada (ao invés de fazer isso no método `inicializar`) e o destrutor apaga os objetos criados. Apesar disso, o uso da **InterfaceSerial** é o mesmo do EP1.

Assim como no EP1, você pode fazer uma classe de teste, mas ao entregar use a classe **InterfaceSerial** passada.

4 Interface com o usuário

A interface com o usuário foi implementada usando o *framework* Qt (https://www.qt.io). Esse *framework* pode ser usado gratuitamente para o desenvolvimento de projetos *open source*, mas é pago para projetos comerciais. Com isso é possível gerar gráficos como o apresentado abaixo.



A instalação do Qt para o Windows e sua configuração no CodeBlocks é apresentada no *Tutorial de Instalação do Qt*, disponível no e-Disciplinas. Note que para fazer o EP não será necessário aprender o Qt e não é obrigatório usá-lo. Caso você não queira usar o Qt, crie uma classe de testes, assim como descrito para a classe **InterfaceSerial** no EP1.

> **Cuidado**: não faça inclusão de classes do Qt nas classes que você desenvolveu. Se você fizer isso, seu código não compilará no Judge. Você deve apenas usar a classe **Tela**.

[1] Uma forma mais elegante de tratar da orientação seria usar um Enumerador (`enum`). Porém, como isso não foi explicado na disciplina, preferiu-se uma solução mais simples.

4.1 Classe Tela

A classe **Tela** é a intermediária entre a sua classe **Grafico** e o Qt. Você não deve alterá-la: é apenas necessário chamar os métodos na ordem adequada. Apenas a classe **Grafico** deve chamar os métodos da **Tela**. Na correção usaremos uma implementação nossa da classe **Tela**, substituindo a classe **Tela** entregue.

A classe **Tela** possui os seguintes métodos e constantes:

```
static const int LARGURA_PADRAO;
static const int ALTURA_PADRAO;

Tela();
virtual ~Tela();

/**
 * Define o eixo X.
 */
virtual void setEixoX(string nome, double minimo, double maximo);

/**
 * Define o eixo Y.
 */
virtual void setEixoY(string nome, double minimo, double maximo);

/**
 * Plota um ponto (x, y) para uma Serie com o nome informado.
 */
virtual void plotar(string nomeDaSerie, double x, double y);

/**
 * Mostra o gráfico, criando uma janela com largura e altura
 * informadas. O método retorna apenas quando a janela é fechada.
 */
virtual void mostrar(int largura, int altura);

/**
 * Mostra o gráfico, criando uma janela com largura e altura padrão.
 * O método retorna apenas quando a janela é fechada.
 */
virtual void mostrar();
```

Para apresentar um gráfico usando a classe **Tela** é necessário definir os eixos (usando os métodos `setEixoX` e `setEixoY`), `plotar` os pontos e só então chamar o método `mostrar`. Não chame o método `mostrar` sem ter os eixos definidos.

Um exemplo simples de uso da classe **Tela** é apresentado no *Tutorial de instalação do Qt*, disponível no e-Disciplinas.

4.2 Método main

Coloque o main em um arquivo em separado, chamado `main.cpp`. O main deve pedir para o usuário as informações do gráfico e então desenhá-lo. Assim como no EP1, **a ordem das mensagens informadas pelo usuário deve seguir exatamente a ordem definida**, assim como o formato das respostas esperadas.

O `main` deve pedir para o usuário o nome da **Serie**. Em seguida ele deve pedir para o usuário o canal da coordenada X. As opções válidas devem ser os canais informados pela **InterfaceSerial**, além do canal "Tempo" que deve ser a opção `0`. O main deve então pedir o canal da coordenada Y, apresentando a lista de canais informada pela **InterfaceSerial** (não coloque o "Tempo"). Em seguida o main deve perguntar o número de pontos que o usuário deseja obter. Com isso, crie um objeto **Serie** ou **SerieTemporal** e

adicione nele os valores obtidos pelos canais escolhidos, através da **InterfaceSerial**. Depois de obter os pontos, pergunte, nesta ordem, se os eixos X e Y do gráfico deve ser estático ou dinâmico. Se o usuário escolher estático, pergunte o título, o mínimo e o máximo; se ele escolher dinâmico, pergunte o mínimo padrão e o máximo padrão. Com essas informações crie um objeto **Grafico** e mostre-o na tela.

> **Atenção:** A interface com o usuário deve seguir exatamente a ordem definida (e exemplificada). Se a ordem não for seguida, haverá desconto de nota.

Um exemplo de entrada e saída é apresentado abaixo, indicando em vermelho os valores informados pelo usuário.

```
Aperte o botao reset da placa.
Informe o nome da serie: Aceleracao
Escolha o canal X:
0) Tempo
1) ACCX
2) ACCY
3) ACCZ
4) MAGX
5) MAGY
6) MAGZ
0
Escolha o canal Y:
1) ACCX
2) ACCY
3) ACCZ
4) MAGX
5) MAGY
6) MAGZ
2
Obter quantos pontos? 5
Obtendo os pontos
Gerando o grafico
O eixo X e estatico ou dinamico (e/d): e
Informe o titulo: Tempo
Valor minimo: 0
Valor maximo: 6
O eixo Y e' estatico ou dinamico (e/d): d
Valor minimo padrao: -0.4
Valor maximo padrao: 0.1
```

5 Entrega

O projeto deverá ser entregue até dia **25/10** no Judge disponível em <https://homologacao.pcs.usp.br/pcs3111/ep/>

A entrega deve ser feita por cada membro da dupla, assim como no EP1 (ou seja, os dois devem submeter o mesmo exercício no Judge do EP). A entrega consiste em três partes. Veja no Judge os detalhes da submissão.

> **Atenção**
> - Deve ser mantida a mesma dupla do EP1. É possível apenas *desfazer* a dupla. Com isso, cada aluno deve fazer uma entrega diferente (e em separado). Caso você deseje fazer isso, envie um e-mail para levy.siqueira@usp.br até dia **18/10**.
> - Os dois membros da dupla devem submeter o EP. Pode haver desconto na nota caso um dos alunos não entregue o EP.

Cada parte deve ser entregue em um arquivo comprimido no formato **zip** (não usem outros formatos). Os fontes não devem ser colocados em pastas. Não submeta arquivos do Qt. O Judge fará uma verificação *básica* do software, verificando se é possível instanciar as classes. Não altere o nome dos arquivos entregues e siga a convenção de nomes para os arquivos ".h" e ".cpp". O não atendimento disso pode levar a erros de compilação (e, consequentemente, nota zero).

O Judge fará uma verificação *básica* do software, verificando se é possível instanciar as classes definidas neste documento e os seus respectivos métodos especificados.

6 Dicas

- Adicione no arquivo de definição das classes **SerieTemporal** e **EixoDinamico** os outros métodos públicos que sejam necessários para que elas sejam filhas de suas classes pais.
- Você pode (e deve, em alguns casos) criar membros privados e protegidos se você quiser. No main você pode criar funções auxiliares – elas podem simplificar bastante o código!
- As classes **Tela** e **InterfaceSerial** devem ser entregues da mesma forma que vocês a receberam. Isso não impede que você faça alterações para testes – é só não entregar com essas alterações.
- Implemente a solução aos poucos – não deixe para implementar tudo no final.