FACULDADE DE TECNOLOGIA DE SÃO PAULO

Linguagem e Técnicas de Programação Orientada a Objetos

1º Trabalho: Classe Data

Este trabalho consiste no desenvolvimento de uma classe de uso geral chamada Data, que deverá representar datas reais. Não se trata de desenvolver um aplicativo para o usuário final, e sim, uma classe que pode ser disponibilizada para uso de vários aplicativos. Assim, temos de estabelecer algumas limitações:

1. Nenhum método dessa classe deve escrever dados na tela. Em vez disso, os dados que seriam escritos na tela devem ser retornados por esse método, e o aplicativo se encarregará da escrita.
2. Nenhum método deverá fazer leitura do teclado. Em vez disso, o aplicativo se encarregará dessa leitura, com uma interface de usuário apropriada, e os dados lidos deverão ser enviados como parâmetros.
3. Não escrever na tela mensagens de erro. Em vez disso, escrever métodos booleanos que retornam true para ok, e false para erro.

Cada objeto Data deverá acionar consistência automática sempre que seu conteúdo for alterado, seja na construção, seja alteração solicitada por um usuário (de um aplicativo que venha a utilizar esta classe). Assim, cada método de instancia que modifica o conteúdo deve ativar a consistência.

A classe deve dar suporte aos seguintes formatos de data:

- Britânico: DD/MM/AAAA
- Americano: MM/DD/AAAA
- Italiano: DD-MM-AAAA
- Germânico: DD.MM.AAAA
- ANSI: AAAA.MM.DD

Obs.: usar ano de 4 dígitos.

Observe que formato de data é simplesmente uma forma de exibição e entrada de dados. Formato de data não muda o conteúdo.

Como gerenciar o formato da data?

Para garantir uma ótima performance dos métodos, o formato da data não deve ser gerenciado da forma intuitiva que muitos de nós pensariam a principio: quando um método estiver prestes a exibir a data na tela, fazer um *switch case* para ver qual o formato. Dessa forma teremos um trecho de programa que se repete em outras partes, e que fatalmente tornará os aplicativos mais lentos. Temos como sugestão a seguinte forma de implementação:

Em vez de declarar variáveis separadas para dia, mês e ano (`int dia, mes, ano;`), vamos criar um vetor de int com 3 elementos chamado, por exemplo, de componentes. Assumiremos que componentes[0] é o dia, componentes[1] é o mês e componentes[2] é o ano. Esse vetor será a única variável de instancia desta classe.

Em seguida, vamos ver o que diferencia um formato de outro. Na verdade, são duas coisas: o caracter separador (/-.) e a ordem dos termos. Nossa classe Data já pode ter 2 variáveis de classe: *formato* e *separador*. São variáveis estáticas com inicialização obrigatória.

Atributo *formato*: deverá assumir um dos seguintes valores: 0 para britânico, 1 para americano, 2 para italiano, 3 para germânico e 4 para ANSI. Entre esse formatos, um deles deve ser escolhido para formato default, então vamos escolher o britânico que é o formato padrão no Brasil. Então teremos:

```
static int formato = 0;
```

A ordem dos termos:

Podemos declarar um vetor estático, vetor int com 3 elementos, chamado de termos, por exemplo. Seu conteúdo será o índice dos termos, da seguinte forma: termos[0] será o índice do 1º termo, isto é, aquele que aparece antes do 1º separador; termos[1] será o índice do 2º termo, isto é, aquele que aparece entre os separadores; e termo[2] é o índice do 3º termo, isto é, aquele que aparece aos o 2º separador. Dessa forma, para o formato britânico o conteúdo do vetor será {0, 1, 2}. Sua declaração:

```
static int termos[] = {0, 1, 2};
```

Montar a string da data:

```
"" + componentes[termos[0]]+separador+componentes[termos[1]]

+separador+componentes[termos[2]]
```

Transformar String em Data:

Encontrar a posição dos separadores, utilizando o método `charAt()` da classe String: `if(strdata.charAt(i) == separador)`, onde i é a posição do caracter, a partir de 0, strdata é um objeto local da classe String.

Supondo que as posições encontradas estão nas variáveis pos1 e pos2, obter as seguintes substrings:

```
sub1 = strdata.substring(0, pos1);

sub2 = strdata.substring(pos1+1, pos2);

sub2 = strdata.substring(pos2+1);
```

Agora, obter o valor numérico e atribuir aos termos da data

```
componentes[termos[0]] = Integer.parseInt(sub1);

componentes[termos[1]] = Integer.parseInt(sub2);

componentes[termos[2]] = Integer.parseInt(sub3);
```

Em seguida ativar o método que faz a consistência.

Consistência da data:

É interessante que o trabalho da consistência seja dividido em alguns métodos, que por sua vez serão úteis para outras tarefas.

Ano bissexto: verificar se o ano é divisível por 4 e não por 100, ou se é divisível por 400. Este método é melhor ser estático:

```
static boolean bissexto(int ano) { ... }
```

Quantos dias tem o mes m:

```
static int diasMes(int m, int a){ ... }
```

Assim, consistencia sera simplesmente verificar mes entre 1 e 12 e verificar se o dia da data não ultrapassa o limite de cada mês. No caso de fevereiro, verificar ano bissexto caso o dia seja 29.

Mudar o formato padrão da data:

A classe deve deixar disponível este método caso seja desejado mudar o formato padrão da data. Trata-se de um método de classe, já que manipula diretamente as variáveis de classe.

```
static boolean mudaFormato(int f) { ...}

// retorna false se o formato for inválido
```

Construtores:

Construtor inicializador 1: `public Data(int d, int m, int a) { ... }`

Construtor inicializador 2: `public Data(String sd) { ... }`

Construtor de cópia: `public Data(Data d) { ... }`

Métodos de Instancia:

```
public boolean stringData(String s) { ... }
```

A instancia que ativa este método deve assumir o valor correspondente à string recebida como parâmetro. A data passará pela consistencia, retornará true para data OK, e false para data inválida.

```
public String dataString() { ... }
```

Retorna uma string a partir do conteúdo da instancia que ativa o método, de modo que seja respeitado o formato.

Métodos auxiliares privativos:

```
private long dataDias() { ... }
```

Fornece um valor `long` que significa o nº de dias decorridos desde 1 de janeiro de 1900 até a data conteúdo da instancia.

```
private void diasData(long d) { ... }
```

A partir de `d`, que representa dias desde 1-jan-1900, define o conteúdo da instancia que ativa o método.

Operações com datas:

Soma: Data + dias => Outra Data posterior em dias

Subtração 1: Data – dias => Data anterior em dias

Subtração 2: Data – Data => dias decorridos entre as duas datas.

Implementar os métodos:

```
public Data soma(int dias) { ... }

public Data sub(int dias) { ... }

public long sub(Data d) { ... }
```

Bom desenvolvimento.