PCS3111

Laboratório de Programação Orientada a Objetos para Engenharia Elétrica

Aula 10: Persistência de objetos

Escola Politécnica da Universidade de São Paulo

Agenda

1. Persistência de informações
2. Persistência em arquivos
3. Acesso a arquivos em C++
4. Tratamento de Erros

Ciclo de vida de um objeto

- Objetos são temporários
  - Fechamento do escopo (alocação estática)
  - `delete` (alocação dinâmica)
  - Fim do programa
- Nem sempre é possível manter todos os objetos em memória

- Como fazer para que um programa *lembre d*os objetos ao executá-lo novamente?
  - Persistência
    - Arquivo (XML, CSV, formato próprio etc.)
    - Banco de dados (várias opções)

Persistência

- O que persistir?
  - Considere objetos da seguinte classe

Aluno
nome: string numeroUSP: int status: int
Aluno(nome: string, numeroUSP: int) getNome() getNumeroUSP() ativar() jubilar() desativar() getStatus() ~Aluno()

Persistência

- Qual a classe mais adequada para fazer a persistência?

- Problemas de usar a própria classe
  - Coesão
    - Responsabilidades do objeto X *Gestão* de objetos
  - Dependência a uma forma de persistência
    - Lógica de persistência misturada à lógica da classe

- Solução
  - Desacoplar a persistência da classe: usar uma classe específica para isso

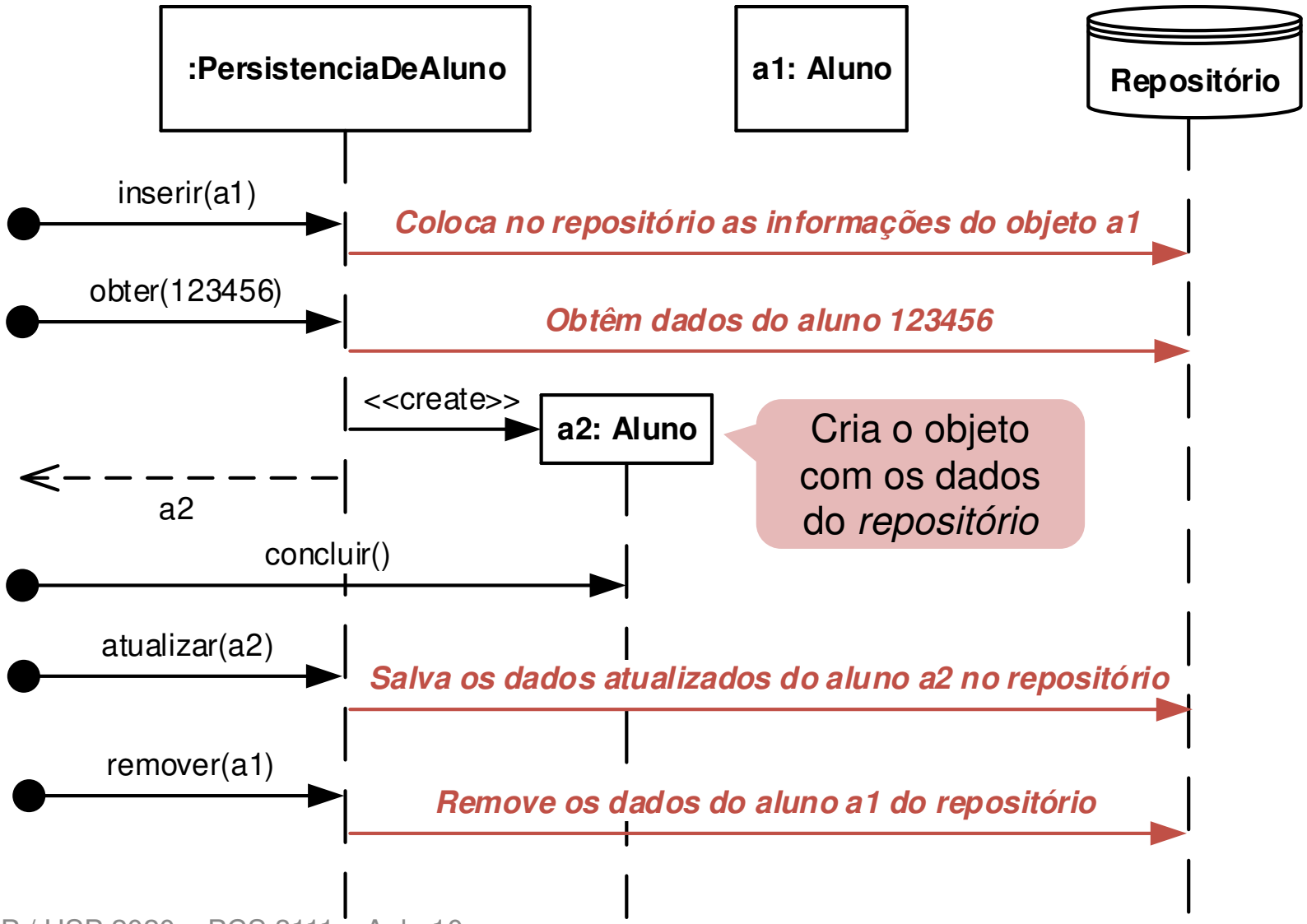
Classe de Persistência

- *Em geral* definem-se os seguintes métodos
  - `inserir (<Objeto>)`
    - Insere um objeto
  - `remover(<Objeto>)` ou `remover(identificador)`
    - Remove um determinado objeto
  - `atualizar(<Objeto>)`
    - Atualiza os dados de um objeto
  - `obter()` ou `obter(dados de pesquisa)`
    - Obtêm *todos* os objetos
    - Obtêm algum objeto específico → pesquisa

Não é obrigatório ter **todos** esses métodos. Deve-se **analisar** o que é necessário.

Classe de Persistência

- Exemplo



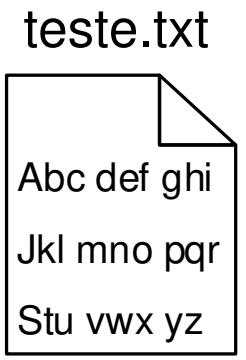
Conclusão

- A forma de persistência deve ser *transparente*
  - Pode ser um formato próprio, XML, CSV, BD, etc.
- Existem vários outros detalhes
  - Formas de melhorar o desempenho
  - Persistir valores de atributos que são *objetos*
  - Persistir objetos de classes que tem classes pais
  - Lidar com vários objetos representando o mesmo objeto persistido
    - De uma forma geral, evite isso!
- Existem bibliotecas que podem cuidar da persistência de objetos

Persistência em arquivos

Arquivo

- Faremos a persistência em arquivo

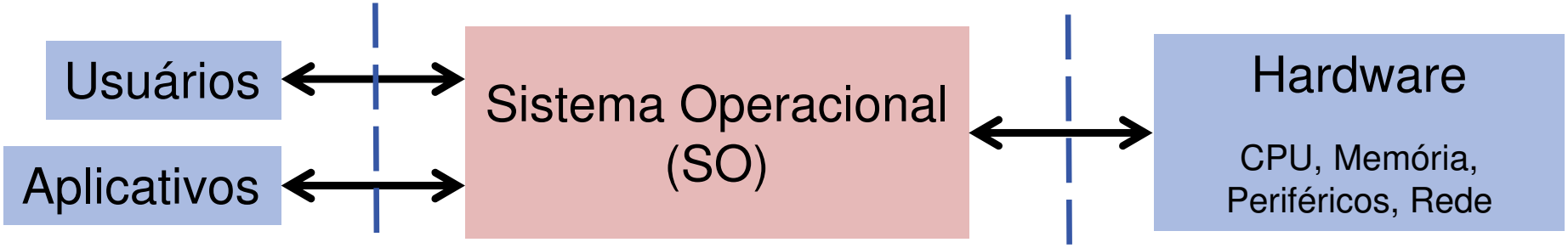


- Arquivo: noção intuitiva
  - Sequência de *bytes*
  - Armazenamento não-volátil
  - Identificados por uma *string* (nome do arquivo)

- São armazenados em um hardware
  - *Exemplo*: disco rígido, drive USB e CD/DVD
  - Os detalhes da manipulação de arquivos são de responsabilidade do sistema operacional

Sistema Operacional

- Camada de software que fornece serviços básicos de forma a unificada a *processos*
  - Obs.: processo ≈ programa em execução



- Disponibiliza uma API para operações sobre arquivos
  - API: *Application Programming Interface*

Sistema Operacional

- Comunicação para o uso de um arquivo
  1. O **processo** requisita ao **SO** a **abertura** de um arquivo
  2. O **SO** abre o arquivo e devolve ao **processo** um **identificador** do arquivo aberto
  3. O **processo** pede ao **SO** uma **operação** (leitura / escrita) utilizando o identificador
  4. O **processo** requisita ao **SO** o **fechamento** do arquivo
     - Importância do fechamento
       - Algumas mudanças podem não ter sido efetuadas *ainda*
       - Número limitado de arquivos que podem ser abertos por um processo
       - É possível permitir que o arquivo só seja usado por um processo por vez

Modo de Leitura e Escrita

- **Texto**
  - Informações codificadas em texto
  - Decodificação automática de caracteres
  - Suporte a leitura de linhas e palavras
  - Compreensível pelo ser humano

- Binário
  - Informações codificadas em *bytes* (similar a memória)
  - Leitura e escrita rápidas
  - Comuns em mídias (imagens, sons, vídeos, etc.)

Persistência de Objetos em Arquivo

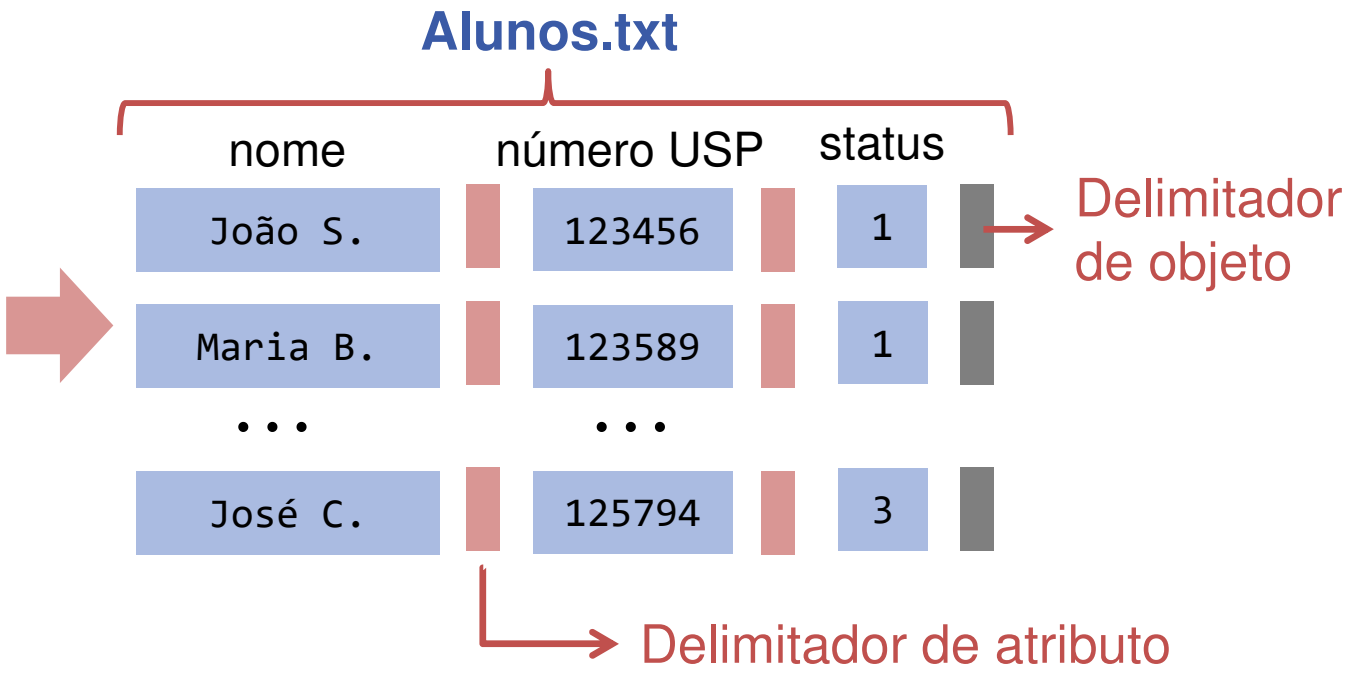
- Para persistir objetos em **arquivo texto** o programador deve manipular o arquivo
  - Escrever os **atributos** relevantes de cada objeto
  - Recriar os objetos a partir dos **dados** escritos

- Necessário definir um **formato** para o arquivo
  - Ordem dos dados
  - Delimitador para os dados
    - Conseguir diferenciar onde termina um dado e começa outro
  - Outras codificações que forem necessárias
    - *Exemplo*: código para dizer qual é o **tipo** da classe

Persistência de Objetos em Arquivo

- *Exemplo*: um vetor de Alunos

Aluno
nome: string numeroUSP: int status: int
Aluno(nome: string, numeroUSP: int) getNome() getNumeroUSP() ativar() jubilar() desativar() getStatus() ~Aluno()



- O que usar como delimitador de objeto ou de atributo?
  - Depende do dado e da facilidade de processar!

Acesso a arquivos em C++

Stream

- Em C++ arquivos são manipulados como **streams**
  - Fonte/destino de dados

- A **entrada e saída padrão** também são streams
  - Necessário `#include <iostream>` e `using namespace std`
    - `cin` → Entrada Padrão (teclado) (lê-se: "c" in)
    - `cout` → Saída Padrão (tela) (lê-se: "c" out)
    - `cerr` → Saída de Erro (lê-se: "c" err)
    - `clog` → Saída de Log (lê-se: "c" log)
  - `cin` é um objeto do tipo `istream`
  - `cout`, `cerr` e `clog` são objetos do tipo `ostream`

Manipulação de arquivo

- Similar à manipulação da entrada/saída padrão
  - Necessário `#include <fstream>`
    - Objetos `ifstream` para leitura (derivados de `istream`)
    - Objetos `ofstream` para escrita (derivados de `ostream`)
  - Existem algumas *pequenas diferenças*
    - Declaração de variável
    - Abertura e fechamento do arquivo
    - Tratamentos de erros de leitura e escrita

EX01

```
 7    ifstream input;
 8    input.open("dados.txt");
 9
10    int x;
12    input >> x;
...
15    input.close();
```

Abertura

```
19    ofstream output;
20    output.open("dados.txt");
21
22    output << 10 << " " << "Teste";
23
24    output.close();
```

Fechamento

Abertura de arquivo

- Formato geral
  - `in_or_out_stream.open(arquivo, parâmetros)`

Opcional

Parâmetros (múltiplos possíveis usando "|")	Significado
`ios_base::app`	(**app**end) escritas no final de arquivo
`ios_base::ate`	(**at e**nd) move para final após abertura
`ios_base::binary`	(**binary**) modo binário
`ios_base::in`	(**in**put) operações de leitura
`ios_base::out`	(**out**put) operações de escrita
`ios_base::trunc`	(**trunc**ate) apaga conteúdo atual do arquivo

  - Os parâmetros `in` e `out` são automáticos para `ifstream` e `ofstream` respectivamente
  - Se o arquivo não existir, ele é criado ao abrir o `ofstream`

Fechamento de Arquivos

- Um arquivo pode ser desconectado de um programa chamando o método `close()`

```
 2  #include <fstream>
 3
 4  using namespace std;
 5
 6  int main() {
 7    ifstream input;
 8    input.open ("dados.txt");
...
15    input.close();
```

```
19    ofstream output;
20    output.open ("dados.txt");
...
24    output.close();
25  }
```

EX01

- Observações
  - **Não usaremos ponteiros** para entrada e saída de arquivos por causa da forma de leitura e escrita
  - O método `close` é chamado automaticamente ao fechar o escopo (pelo destrutor)

Leitura e Escrita

- Usa-se o >> e <<, como no `cin` e `cout`

```
 7    ifstream input;
 8    input.open ("dados.txt");
 9
10    int x;
11    input >> x;
12    string y;
13    input >> y;
14
15    input.close();
```

```
19    ofstream output;
20    output.open ("dados.txt");
21
22    output << 10 << " " << "Teste";
23
24    output.close();
```

EX01

Colocando espaço como separador

- Cuidados
  - O padrão usa espaço/tab/enter como delimitador para leitura
    - Se for necessário ler uma string que tem espaço, pode-se usar o `getline` (vide material extra da Aula 5)
  - Na escrita não é colocado nenhum separador por padrão

Observações

- O local do arquivo pode ser indicado usando caminho (*path*) **absoluto** ou **relativo**
  - **Absoluto**: endereço completo
    - Exemplo (windows): `"C:\\PCS3111\\Aula10\\Teste.txt"`
  - **Relativo**: endereço a partir do local de execução do programa
    - Exemplo: `"Teste.txt"`
      - É o arquivo “Teste.txt” na pasta em que se está executando o programa.

Tratamento de erros

Tratamento de Erros

- Erros ao se trabalhar com arquivos
  - Arquivo não existe
  - Fim de arquivo
  - Erro na leitura
    - *Exemplo*: string quando esperava inteiro
  - Erro no dispositivo

Tratamento de Erros

- Baseado na verificação de estado
  - *Muito antes de C++ incorporar exceções*
- Estado composto por 4 bits (mais de um *bit* pode estar ativo)
  - *goodbit*: sem erros
    - (Qualquer um dos demais bits é considerado erro)
  - *eofbit*: fim de arquivo encontrado em operações de leitura
  - *failbit*: falha na leitura de um valor; erro lógico
  - *badbit*: erro no dispositivo de E/S

Dica: pode-se recuperar de um *failbit* mas não de um *badbit*

Tratamento de Erros

- Métodos auxiliares

	`good()`	`eof()`	`fail()`	`bad()`
goodbit == 1	1	0	0	0
eofbit == 1	0	1	X	X
failbit == 1	0	X	1	X
badbit == 1	0	X	1	1

  - Observações
    - Conversão automática para *`bool`* ajuda a verificar erros
    - A variável `ifstream` pode ser usada diretamente para verificar se `failbit == 0 && badbit == 0`
    - EOF **depende da operação feita no stream** (e *não da próxima*)
      - Se há um `\n` depois do último valor, o *eofbit* ainda será 0; ao tentar ler o *eofbit* e o *failbit* ficarão em 1

Exemplo – Média

```
 7  ifstream entrada;
 8  entrada.open ("numeros.txt");
 9
10  if (entrada.fail() ) {
11    cout << "Arquivo nao encontrado"
           << endl;
12    entrada.close();
13    return 1;
14  }
15
16  double x, total = 0;
17  int quantidade = 0;
18
19  entrada >> x;
20  while (entrada) {
21    total = total + x;
22    quantidade++;
23    entrada >> x;
24  }
25
26  if (!entrada.eof()) {
27    cout << "Erro de leitura" << endl;
28    entrada.close();
29    return 1;
30  }
```

Enquanto não for nem *fail* e nem *bad*

Se saiu do laço sem chegar no fim do arquivo

```
32  if (quantidade == 0)  {
33    cout << "Arquivo vazio" << endl;
34    entrada.close();
35    return 1;
36  }
37
38  double media = total / quantidade;
39  entrada.close();
40
41  ofstream saida;
42  saida.open ("media.txt");
43
44  if (saida.fail())
45    cout << "Erro ao escrever" << endl;
46  else
47    saida << media;
48
49  saida.close();
```

EX02

Detalhes

- O bit de erro não é *reiniciado* para uma nova leitura: uma vez ativo, ele se mantém ativo
  - Operações **não são executadas** se o *stream* não estiver *good*

EX03

```
16    int a;
17    string b = "123";
18    entrada >> a; // ERRO
19    if (entrada.fail())
20      cout << "Erro: nao eh int" << endl;
21
22    entrada >> b;
23    if (entrada.fail())
24      cout << "Erro: " << b << endl;
```

Como está em *fail*, b não é alterado

Arquivo

```
abc
def
```

Saída

```
Erro: nao eh int
Erro: 123
```

  - Na prática, não tem problema em fazer

```
cin >> a >> b >> c;
```

Boas Práticas

- Na abertura de arquivos, verificar se ela foi bem sucedida antes de usar o arquivo
  - Para entrada ou saída
  - Ex.: usar o `fail()`

- Para verificar se o arquivo existe antes de escrever, deve-se tentar ler o arquivo antes
  - Se não for verificada e o arquivo existir, os dados antigos são sobrescritos com dados novos
    - (Por padrão, mas com o parâmetro `ios_base::app` escreve-se no final)

Bibliografia

- LAFORE, R. **Object-Oriented Programming in C++**. Sams, 4th ed. 2002.
- SAVITCH, W. **C++ Absoluto**. Pearson, 1st ed. 2003.
- SUN MICROSYSTEMS. **Core J2EE Patterns - Data Access Object**. 2001. Disponível em: <http://www.oracle.com/technetwork/java/dataaccessobject-138824.html>.