Estrutura de dados - Pilhas (continuação)

Prof. Leonardo Cabral da Rocha Soares
leonardo.soares@newtonpaiva.br
Centro Universitário Newton Paiva

16 de maio de 2019

Na aula de hoje:

- Aula passada
  - Definição de pilhas
  - Operações
  - Formas de implementação
  - Complexidade
- Implementações
  - Por arranjo
  - Por referência
  - Utilizando Java Collections Framework
- Exercícios

Avisos

Pilhas

Descrição

- Uma pilha é um contêiner de objetos que são inseridos e retirados de acordo com o princípio de que o **último que entra é o primeiro que sai** (do inglês, Last In First Out, LIFO).
- O nome pilha deriva-se da metáfora de uma pilha de pratos em uma cantina.
- As pilhas são estruturas de dados fundamentais sendo utilizadas em muitas aplicações, por exemplo:

Pilhas - Exemplos

- Navegadores *web* armazenam os endereços mais recentemente visitados em uma pilha. Cada vez que o navegador visita um novo site, o endereço do site é armazenado na pilha de endereços (*push*). Usando a operação de retorno, *back*, o navegador permite que o usuário retorne ao último site visitado, retirando o endereço do topo da pilha (*pop*).
- Editores de texto geralmente oferecem um mecanismo de reversão de operações (*undo*) que cancela as operações recentes e reverte um documento ao estado anterior à operação. O mecanismo de reversão é implementado mantendo-se as alterações no texto em uma pilha.

Operações

O tipo abstrato de dados (TAD) pilha deve, obrigatoriamente, suportar os métodos:

- push(o): Insere o objeto o no topo da pilha.
- pop(): Retira o objeto no topo da pilha e o retorna; se a pilha estiver vazia, ocorre um erro.

Adicionalmente, podemos definir os seguintes métodos auxiliares:

- tamanho(): Retorna o número de objetos na pilha.
- vazia(): Retorna um *boolean* indicando se a pilha está vazia.

Formas de implementação

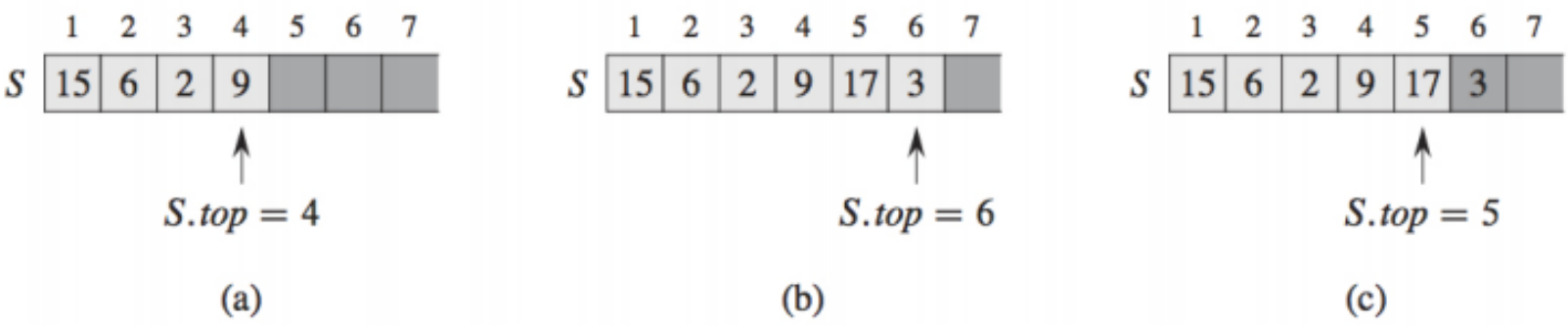
Existem várias opções de estruturas de dados que podem ser usadas para representar pilhas. As duas representações mais utilizadas são:

- Por meio de arranjos.
- Por meio de referências.

Independente da forma de implementação, uma pilha é uma lista com restrições quanto às formas de inserção e remoção, o que permite a reusabilidade de código.

Implementação por arranjos

Os itens da pilha são armazenados em posições contíguas de memória.
Como as inserções e as remoções ocorrem no topo da pilha, um campo chamado **topo** é utilizado para controlar a posição do item no topo da pilha.



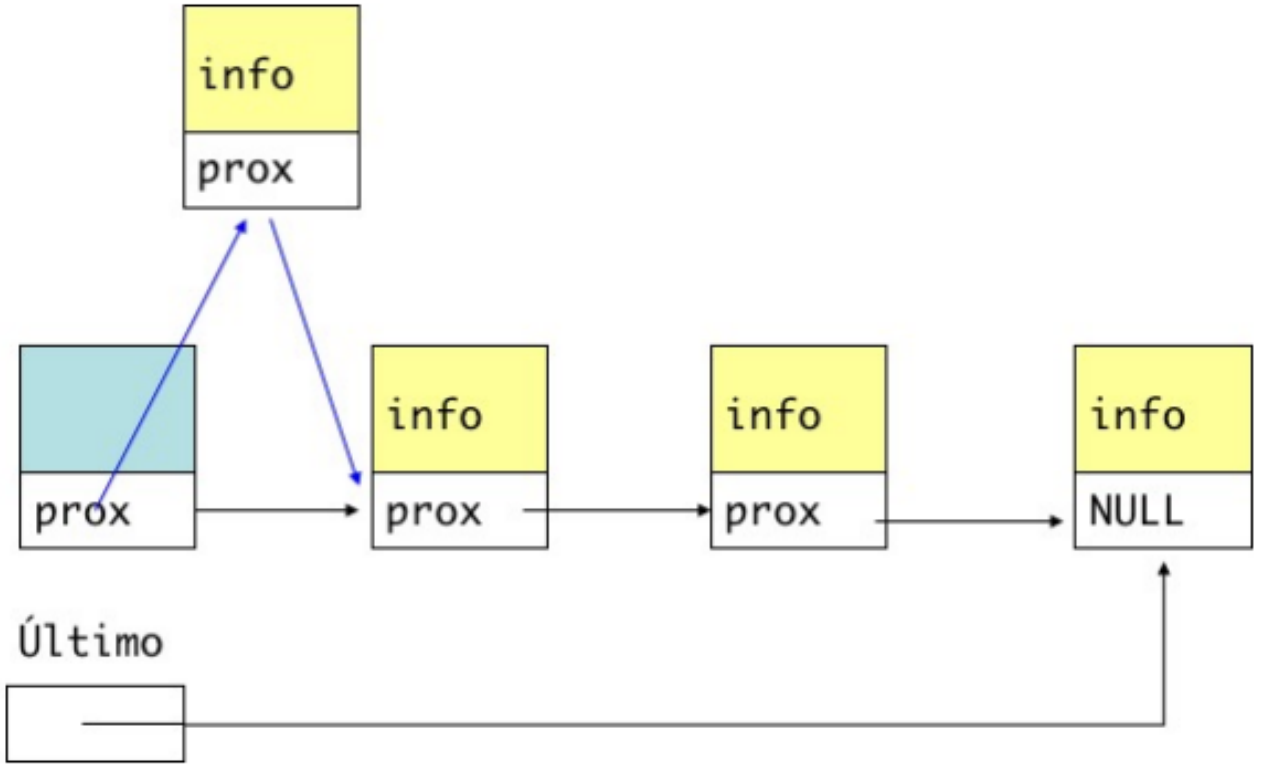
(a) Uma pilha com quatro elementos. (b) Após empilhar (push) dois elementos. (c) Após desempilhar (pop) um elemento.

Implementação por referência

- Cada célula de uma pilha contém um item da pilha e um apontador para outra célula.
- A estrutura contém um apontador para o topo da pilha (célula cabeça).
- Criar um campo tamanho evita a contagem do número de itens na função tamanho.

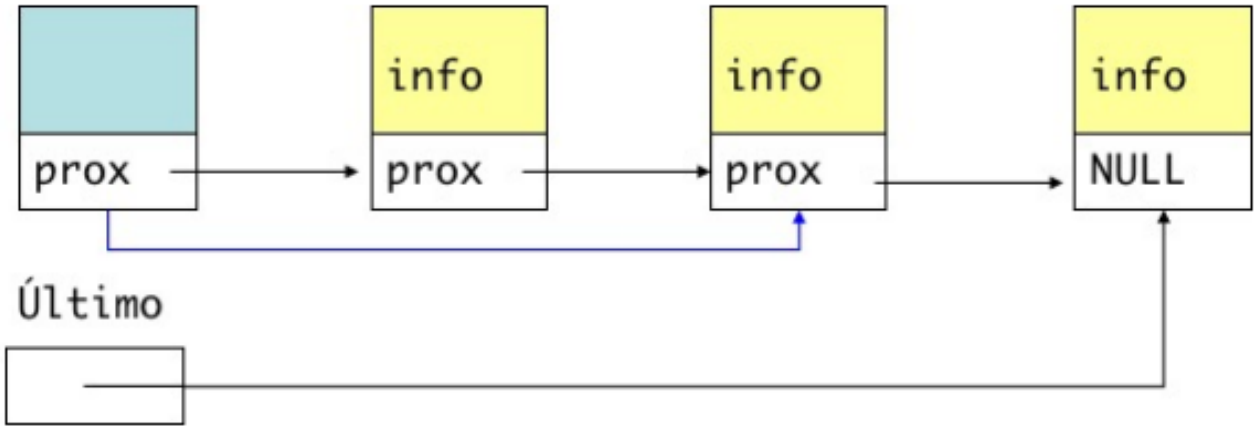
Implementação por referência - Inserção

De acordo com a política **LIFO**, há apenas uma opção de posição onde podemos inserir elementos: o topo da pilha (ou seja, primeira posição).



Implementação por referência - Remoção

De acordo com a política **LIFO**, há apenas uma opção de posição onde podemos remover elementos: o topo da pilha (ou seja, primeira posição).



Complexidade

A complexidade de todas as operações é mantida da implementação de Lista:

- Empilhar: $\theta(1)$
- Desempilhar: $\theta(1)$

Perguntas?

Implementação por arranjo

```java
public class Processo {
    int codigo;
    String responsavel;
    String cliente;
    public Processo(int codigo, String responsavel, String cliente) {
        this.codigo = codigo;
        this.responsavel = responsavel;
        this.cliente = cliente;
    }
    public Processo() {
    }
}
```

```java
public class Pilha {
    static final int MAX_TAM = 100;
    Processo[] pilha = new Processo[MAX_TAM];
    int topo = -1;
    public boolean isVazia(){
        return topo == -1;
    }
    public int getTamanho(){
        return topo+1;
    }
    public void push(Processo p) throws Exception{
        if (topo==MAX_TAM-1)
            throw new Exception ("Não há espaço disponível");
        pilha[++topo] = p;
    }
    public Processo pop() throws Exception{
        if (isVazia())
            throw new Exception ("Lista vazia");
        return pilha[topo--];
        // Atenção ao operador de pós-decremento
    }
}
```

```
public class Main {

    public static void main(String[] args) throws Exception{
        Pilha p = new Pilha();
        Processo proc = new Processo();
        p.push(new Processo(1, "Rosimeire", "Acme"));
        p.push(new Processo(2, "Afonso", "Sansung"));
        p.push(new Processo(3, "Rosimeire", "Lenovo"));
        p.push(new Processo(4, "Ana", "Lenovo"));
        p.push(new Processo(5, "Afonso", "Acme"));
        p.push(new Processo(6, "Rosimeire", "Lenovo"));

        System.out.println("Lista de processos a serem executados:");
        while (!p.isVazia()){
            proc = p.pop();
            System.out.printf("Responsável: %s\t\t Código: %d\tCliente: %s\n",
                    proc.responsavel, proc.codigo, proc.cliente);
        }
    }

}
```

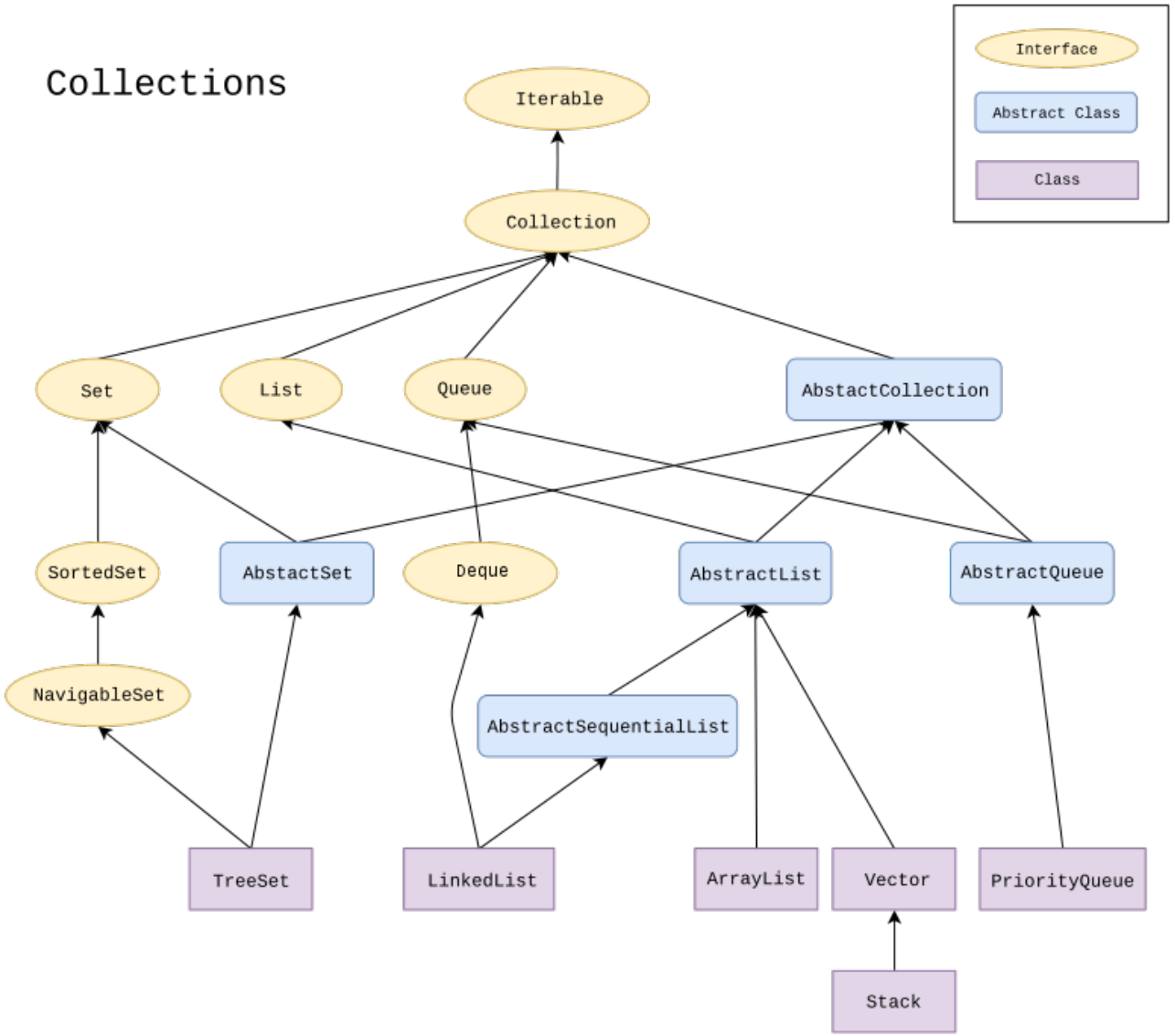
Implementação por referência

```java
public class Processo {
    int codigo;
    String responsavel;
    String cliente;
    Processo prox;

    public Processo(int codigo, String responsavel, String cliente) {
        this.codigo = codigo;
        this.responsavel = responsavel;
        this.cliente = cliente;
        this.prox = null;
    }

}
```

```java
public class PilhaReferencia {
    int tam = 0;
    Processo topo;
    public boolean isVazia(){
        return tam==0;
    }
    public int getTamanho(){
        return tam;
    }
    public void push(Processo p){
        if (tam!=0)
            p.prox = topo;
        topo = p;
        ++tam;
    }
    public Processo pop() throws Exception{
        if (tam==0)
            throw new Exception ("Pilha vazia");
        Processo p = topo;
        topo = topo.prox;
        --tam;
        return p;
    }
}
```

```java
public class Main {
    public static void main(String []args) throws Exception{
        //PilhaArranjo pilha = new PilhaArranjo();
        PilhaReferencia pilha = new PilhaReferencia();
        Processo proc;
        pilha.push(new Processo(1,"Rosimeire","Acme"));
        pilha.push(new Processo(2,"Afonso","Samsung"));
        pilha.push(new Processo(3,"Rosimeire","Lenovo"));
        pilha.push(new Processo(4,"Ana","Lenovo"));
        pilha.push(new Processo(5,"Afonso","Acme"));
        pilha.push(new Processo(6,"Rosimeire","Samsung"));

        System.out.println("Lista de processos a serem executados");
        while(!pilha.isVazia()){
           proc = pilha.pop();
           System.out.printf("Código: %d\t\tResponsável: %s\t\tCliente: %s\n",
                   proc.codigo,proc.responsavel,proc.cliente);
        }
    }
}
```

Utilizando Java Collections Framework



```java
public class Processo {
    int codigo;
    String responsavel;
    String cliente;
    public Processo(int codigo, String responsavel, String cliente) {
        this.codigo = codigo;
        this.responsavel = responsavel;
        this.cliente = cliente;
    }
    public Processo() {
    }
}
```

```
import java.util.Stack;
public class Main {
    public static void main(String []args) throws Exception{
        //PilhaArranjo pilha = new PilhaArranjo();
        //PilhaReferencia pilha = new PilhaReferencia();
        Stack<Processo> pilha = new Stack();

        Processo proc;
        pilha.push(new Processo(1,"Rosimeire","Acme"));
        pilha.push(new Processo(2,"Afonso","Samsung"));
        pilha.push(new Processo(3,"Rosimeire","Lenovo"));
        pilha.push(new Processo(4,"Ana","Lenovo"));
        pilha.push(new Processo(5,"Afonso","Acme"));
        pilha.push(new Processo(6,"Rosimeire","Samsung"));

        System.out.println("Lista de processos a serem executados");
        while(!pilha.isEmpty()){
           proc = pilha.pop();
           System.out.printf("Código: %d\t\tResponsável: %s\t\tCliente: %s\n",
                   proc.codigo,proc.responsavel,proc.cliente);
        }
    }
}
```

1. Descreva a saída da seguinte sequência de operações sobre uma pilha:
   push(5), push(3), pop(), push(2), push(8), pop(), pop(), push(9), push(1), pop(), push(7), push(6), pop(), pop(), push(4), pop(), pop().
2. Implemente um pequeno **jogo de 21** para dois jogadores. A cada jogada, é sorteada uma carta $\in [1, 2, ..., 13]$. O jogador pode pegar quantas cartas desejar e sua pontuação é igual a soma dos valores de suas cartas. Caso a pontuação ultrapasse 21, o jogador poderá descartar a última carta comprada mas isso gerará um desconto de 5 pontos em sua pontuação total. O jogador 2 só joga após o jogador 1 encerrar suas jogadas. Ao final, vence o jogo quem estiver mais próximo de 21 pontos. Exiba as cartas do jogador vencedor.


рахмат
danke
謝謝
spas
ngiyabonga
teşekkür ederim
Баярлалаа
mersi
kia ora
barka
welalin
tack
mahalo
tapadh leat
спасибо
faafetai lava
vinaka
спасибі
blagodaram
dank je
misaotra
matondo
paldies
grazzi
kiitos
dankie
nanni
nandri
dhanyavad
thank you
gracias
хвала
asante
manana
obrigada
tenki
murakoze
bedankt
bayarlalaa
gracie
hvala
mauruuru
köszönöm
dankon
aciu
akun
enkosi
djiere dieuf
tau
mochchakkeram
дякую
mamnun
chokrane
dziękuję
chnorakaloutioun
gratias ago
gràcies
sulpay
taiku
go raibh maith agat
sobodi
děkuji
obrigado
sağolun
sukriya
kop khun krap
ありがとう
tanemirt
grazie
arigatô
takk
dakujem
trugarez
mèsi
didi madloba
najis tuke
terima kasih
rahmet
ευχαριστώ
diolch
dhanyavadagalu
shukriya
merce
мерси
kam sah hamnida
rahmat
তোমাকে ধন্যবাদ
감사합니다
xièxie
merci

Referências

- CARVALHO, Marco Antonio Moreira de. **Projeto e análise de algoritmos**. 01 mar. 2018, 15 jun. 2018. Notas de Aula. PPGCC. UFOP
- GOODRICH, Michael T.; TAMASSIA, Roberto. **Estruturas de Dados & Algoritmos em Java.** Bookman Editora, 2013.
- ZIVIANI, Nivio. **Projeto de Algoritmos com implementações em Java e C++**, 2007.