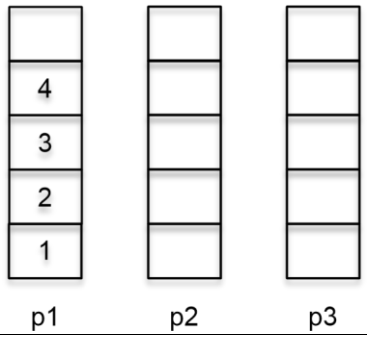
Exercícios: Pilhas

1. Dado o estado inicial das pilhas **`p1`**, **`p2`** e **`p3`** na figura abaixo, mostre o estado final dessas mesmas pilhas após as operações descritas no código apresentado.

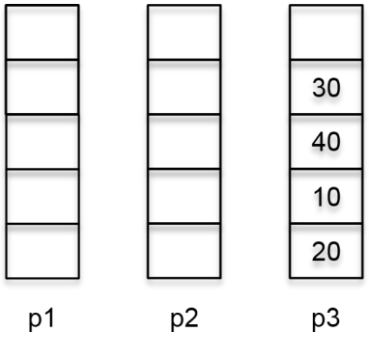



```
try {
      int temp = p1.pop();
      p2.push(temp);
      p3.push(p1.pop());
      p2.push(p1.pop());
      temp = p1.pop();
      p3.push(temp);
      p1.push(p2.pop());
      p3.push(p2.pop());
      p3.push(p1.pop());
} catch (Exception e) {
      System.out.println(e.toString());
}
```

2. Considere o código a seguir, onde a classe **`MinhaPilha`** implementa a interface **`Stack`**:

```
public class Teste {
  public static void main(String args[]) {
    MinhaPilha<Integer> p1 = new MinhaPilha<>();
    MinhaPilha<Integer> p2 = new MinhaPilha<>();
    MinhaPilha<Integer> p3 = new MinhaPilha<>();
    try {
      p1.push(10);
      p1.push(20);
      p1.push(30);
      p1.push(40);
    } catch (Exception e) {
      System.out.println(e);
    }
  }
}
```

Complete o código para que as pilhas fiquem com o seguinte estado final:



3. Implemente em uma classe qualquer o seguinte método:

```
public Integer[] itemsExcept(int number, Stack<Integer> p)
```

Esse método deve percorrer a pilha **`p`** e retornar um vetor com os elementos de **`p`** sem a ocorrência do elemento **`number`**. O conteúdo original da pilha deve ser preservado.

4. Implemente um método que recebe duas pilhas **`s1`** e **`s2`** e transfere os elementos da primeira para a segunda de modo que os elementos em **`s2`** fiquem na mesma ordem que em **`s1`**. Dica: use uma pilha auxiliar.

```
public void transferElements(Stack<E> s1, Stack<E> s2)
```

5. Implemente em uma classe qualquer o seguinte método:

```
void prependStack(Stack<Integer> p1, Stack<Integer> p2)
```

Esse método deve armazenar todos os elementos de **`p2`** em **`p1`** de maneira que eles fiquem abaixo dos elementos originais de **`p1`**, mantendo os dois conjuntos de elementos em sua ordem original. Podem ser utilizados vetores ou pilhas auxiliares.

6. Considere que os seguintes métodos fazem parte da implementação de uma pilha que armazena inteiros:

```
public void add() {
   push(pop() + pop());
}

public void sub() {
   push(-1 * pop() + pop());
}

public void mul() {
   push(pop() *pop());
}

public void moo() {
   int x = pop();
   int y = pop();
   push(y);
   push(x);
}
```

Assinale V ou F para as sentenças a seguir:

( ) A sequência *push(4); push(2); push(2); mul(); moo()* resulta no valor 4 no topo da pilha.

( ) O método **`moo()`** inverte os dois elementos do topo da pilha.

( ) Supondo a pilha inicialmente vazia, a execução da sequência *push(2); push(5); push(3); mul(); push(1); add()* deixa a pilha com um total de 2 elementos.

( ) A sequência *push(8); push(5); sub()* resulta no valor -3 no topo da pilha.

( ) A expressão matemática *5 - 2 ×3* pode ser calculada pela sequência *push(5); push(2); push(3); mul(); sub()*.

7. Suponha que uma sequência de operações *push* e *pop* é realizada em uma pilha. As operações *push* inserem, em ordem, número inteiros de 0 a 9. As operações *pop*, além de retirar o elemento do topo da pilha, exibem o valor desse elemento. Dentre as saídas abaixo, determine aquelas que são possíveis. Por exemplo, a saída "1 2 0" é possível, podendo ser produzida pela sequência *push(0); push(1); pop(); push(2); pop(); pop()*. Observe que os números não precisam ser inseridos todos de uma única vez na pilha.

a) 4 3 2 1 0 9 8 7 6 5
b) 4 6 8 7 5 3 2 9 0 1
c) 2 5 6 7 4 8 9 3 1 0
d) 4 3 2 1 0 5 6 7 8 9

8. Implemente um método que recebe uma pilha como parâmetro e inverte a ordem dos seus elementos. Use somente outras pilhas como estruturas auxiliares.

9. Escreva um algoritmo para verificar se um dado elemento está presente em uma pilha. Em caso positivo, o algoritmo deve fornecer também a posição do item na pilha, considerando a base como posição 0. A pilha deve permanecer a mesma após a execução do procedimento.

Nos exercícios a seguir, os métodos solicitados devem ser implementados dentro das classes que implementam pilhas. Salvo indicação em contrário, as estruturas passadas como parâmetro (arrays, pilhas, etc.) devem ser preservadas, ou seja, seus elementos não devem ser removidos ou trocados de ordem.

10. Implemente o método **`contains`**, definido abaixo, que informa se a pilha contém determinado elemento.

```
public boolean contains(E element)
```

11. Implemente um método que inverte a ordem dos elementos da pilha.

```
public void flip()
```

12. Implemente uma sobrecarga do método **`push`** que recebe como parâmetro uma pilha, em vez de um elemento. Esse método deve adicionar à pilha corrente os elementos da pilha passada como parâmetro, porém mantendo a ordem original — ou seja, o elemento do topo da pilha passada como parâmetro deve ficar no topo da pilha corrente.

13. Implemente um método **`equals`** para a pilha. Uma pilha será igual a outra se contiver os mesmos elementos, empilhados na mesma ordem. Para comparar os elementos, use também o método **`equals`**.

14. Implemente um método **`clone`** para a pilha. Esse método deve retornar uma nova pilha contendo os mesmos elementos da atual. Os elementos em si não devem ser duplicados.