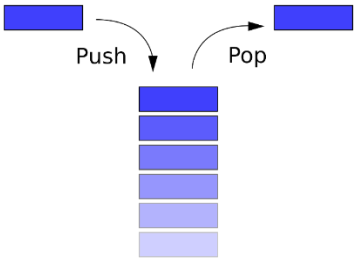
Exercício de Especificação de testes

1) Especificar testes para uma pilha (estrutura de dados)

Nesse exercício vamos especificar um conjunto de 10 casos de teste, com objetivos variados, para verificar a correta implementação de uma Pilha.

Para quem não se lembra, uma Pilha é uma estrutura de dados muito utilizada na construção de sistemas operacionais, algoritmos, etc. É uma estrutura do tipo LIFO (*Last In First Out*), ou seja, o último elemento incluído na Pilha, será o primeiro a sair. Uma aplicação clássica são as pilhas de chamadas de procedimentos (Stack Traces). Mais detalhes podem ser obtidos nesse link: http://www.ime.usp.br/~pf/algoritmos/aulas/pilha.html



Para esse exercício, vamos ignorar a lógica interna (implementação) da pilha e tratar a especificação dos testes como de caixa preta. A nossa pilha terá APENAS os seguintes métodos:

- void Inicializar(int tamanho)
  - Inicializa a pilha com um tamanho fixo. Só cabem a quantidade definida ao inicializar e essa quantidade não pode ser mudada.
- boolean vazia()
  - retorna true se a pilha está vazia (sem nenhum elemento) e false caso contrário.
- boolean cheia()
  - retorna true se a pilha está cheia (com quantidade de elementos igual ao seu tamanho da inicialização) e false caso contrário.
- boolean empilhar(int elemento)
  - Coloca o elemento na pilha e retorna true caso tenha conseguido realizar a operação
- int desempilhar()
  - tira o último elemento inserido da pilha e o retorna na função.

Os casos de teste devem ter um objetivo e uma sequência de passos. Cada passo pode ser uma ação (uma chamada a um dos métodos acima) ou uma verificação (uma comparação de um valor esperado com o valor retornado). O exemplo abaixo deve servir de orientação.

Caso de teste de exemplo 1

Objetivo: testar a correta ordem de desempilhamento de elementos.

Passos:

1. inicializar(3) //inicializa com tamanho 3
2. empilhar(1) //empilha o elemento 1
3. empilhar(2)
4. empilhar(3)
5. e1 = desempilhar()
6. e2 = desempilhar()
7. e3 = desempilhar()
8. verificar e1 == 3
9. verificar e2 == 2
10. verificar e3 == 1 // o desempilhamento é na ordem inversa do empilhamento.

Observações importantes para elaboração dos casos de teste:

- Não inventar métodos que não existem.
- Escreva os passos usando pseudo-código.
- Nas próximas aulas aproveitaremos o resultado desse exercício. Portanto, capriche.
- Não escrever vários casos de teste com objetivos similares, mudando apenas um tamanho. Por exemplo, não vale escrever dois casos de teste como esse acima mudando apenas a quantidade de elementos.
- Assumir que as funções lançam exceções quando são chamadas com parâmetros não esperados. Por exemplo, inicializar(“texto”) lançará uma exceção.

2) Especificar testes para uma função de cálculo de desconto

Nesse exercício iremos especificar 5 casos de teste (sem contar o exemplo) para a função abaixo:

```
int calculaDescontoNaCompra( CategoriaProduto cp, int qtd)
```

A especificação da interface desse método está abaixo:

- CatgoriaProduto pode ser A ou B e Qtd pode variar de 1 a 1000.
- A função faz o cálculo do desconto de acordo com as seguintes regras:
- Categoria A não recebem desconto se nº de itens comprados for inferior a 10; recebem 5% desconto para compras entre 10 e 99 itens; 10% de desconto acima de 100 itens.
- Categoria B recebem 5% de desconto para compras abaixo de 10 itens; 15% de desconto entre 10 e 99 itens; 25% de desconto acima de 100 itens.

Note que como estamos testando apenas um método dessa classe, a descrição dos casos de teste será mais simples, apenas com entradas e saídas. Aqui a ideia é criar várias opções de entrada e saída que exercitem um objetivo comum. Por exemplo:

> Caso de teste de exemplo 1
>
> Objetivo: testar os limites de quantidade para obter descontos
>
> Entradas e saídas esperadas (entrada1, entrada2) -> (saída):
>
> 1. (A, 9) -> 0%
> 2. (A, 10) -> 5%
> 3. (A, 99) -> 5%
> 4. (A, 100) -> 10%
> 5. (A, 1000) -> 10%