Aula 01: 06/03/2017

Tópicos

- apresentação de MAC0323
- tipos abstratos de dados (ADTs)
- API, Interfaces
- implementação
- iteradores
- lista ligadas

MAC0323

Bibliografia principal

- Livro *Algorithms* de Sedgewick & Wayne. Website do livro: `http://algs4.cs.princeton.edu`
- página Estrutura de Dados de Paulo Feofiloff `https://www.ime.usp.br/~pf/estruturas-de-dados/` notas de aula baseadas no livro *Algorithms* de Sedgewick & Wayne

Introdução (por Paulo Feofiloff)

Estruturas de dados servem para organizar os dados de um problema de modo que eles possam ser processados mais eficientemente.

Eficiência tem a ver com escalabilidade: como o tempo de processamento cresce quando a quantidade de dados (ou seja, o tamanho) do problema aumenta? O tempo cresce de maneira moderada? Cresce de maneira explosiva?

Exemplo: Qual a diferença entre encontrar uma determinada palavra em uma lista de 10 palavras e encontrar a palavra em uma lista de 10000 palavras? O tempo de processamento é apenas dez vezes maior? É mil vezes maior? Ou é um milhão de vezes maior?

Biblioteca de programas

Veja a página de Paulo Feofiloff.

Instale o ambiente de programação do livro no seu computador.

Instalar java 8 (eu instalei o Open Programas que utilizaremos:

- `javac`: compilador java
- `java`: máquina virtual do java

```
/usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java
java: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked,
interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=e685545718f8
```

- `checkstyle`: encontra os erros de estilo nos arquivos `.java`
- `findbugs`: encontra os eventuais erros nos arquivos `.class`
- `DrJava`: IDE extremamente leve que é usada para escrever programas em Java. Foi projetada especialmente para estudantes e possui uma interface intuitiva e a habilidade de executar código Java interativamente.

Tipos abstratos de dados (ADTs)

Resumo:

```
- ADT
- API
- implementações de uma ADT
- cliente / interface / implementação
- classes, instâncias, objetos, métodos
```

TAD = tipo abstrato de dados = ADT = abstract data type.

Um **ADT** é um conjunto de coisas, que chamaremos itens, e um conjunto operações sobre essas coisas.

As operações são especificadas em uma interface que chamamos API (*applications programming interface*). A interface diz o que cada operação faz, sem dizer como faz o que faz.

Exemplo de ADT: pilha. As operações são: inserção de um item e remoção de um item (de acordo com certas regras).

Outros exemplos de ADTs: fila, string, fila priorizada, tabela de símbolos, grafo, etc.

Um ADT pode ser implementado (ou seja, concretamente realizado) de diversas maneiras. (Uma pilha, por exemplo, pode ser implementada em um vetor ou em uma lista ligada.) Cada implementação tem suas vantagens e desvantagens (por exemplo, algumas operações ficam mais rápidas enquanto outras ficam mais lentas). É praticamente impossível fazer uma implementação universal, boa para todos os usuários.

Um usuário do ADT é chamado cliente. Um cliente só deve usar as operações especificadas na API e não deve ter acesso aos detalhes da implementação da ADT.

Nesse esquema cliente / interface / implementação, a interface funciona como um contrato entre o cliente e a implementação.

Anatomia de um programa em Java

```java
public class MinhaClass {
    atributos de estado
    variáveis

    // contrutor
    public MinhaClasse(...) {

    }

    // Métodos publicos que fazem parte da API
```

```
    // Métodos privados


    // unit test
    public static void main(String[] args) {

    }
}
```

Listas ligadas

```
    private class Node {
        private Item item;
        private Node next;
    }
```

Iteradores

Em ciência da computação um **iterador** é um mecanismo que permite que percorramos os elementos de um ADT.

Bags

Uma **saco** (= *bag*) é um ADT que consiste em uma coleção de coisas munida de duas operações: `add`, que insere uma coisa na coleção, e `iterate`, que percorre as coisas da coleção (ou seja, examina as coisas uma a uma). A ordem em que o iterador percorre as coisas não é especificada e está fora do controle do cliente.

Primeira implementação

API

`public class`	`BagInteger`	
	`Bag()`	cria um saco de Inteiros vazio
`void`	`add(Integer item)`	coloca item neste saco
`boolean`	`isEmpty()`	este saco está vazio?
`int`	`size()`	número de Inteitos neste saco
`void`	`startIterator()`	inicializa o iterador
`boolean`	`hasNext()`	há itens a serem iterados?
`Integer`	`next()`	próximo item

Cliente:

```
public class Cliente {
    public static void main(String args[]) {
        BagInteger bag = null;
        bag = new BagInteger();
        for (int i=10; i < 20; i++) {
            bag.add(i);
        }
        StdOut.println(bag.size());
        bag.startIterator();
        while (bag.hasNext()) {
            Integer valor = bag.next();
            StdOut.println("Item " + valor);
        }
    }
}
```

Implementação

```
public class BagInteger {

    private Node first;
    private int  n;
    private Node current;

    private class Node {
        private Integer item;
        private Node next;
    }

    public BagInteger() {  // construtor
        first = null;
    }

    public void add(Integer item) {
        Node oldfirst = first;
        first = new Node();
        first.item = item;
        first.next = oldfirst;
        n++;
    }

    public int size() {
        return n;
    }

    public boolean isEmpty() {
        return n == 0;
```

```
    }

    public void startIterator() {
        current = first;
    }

    public boolean hasNext(){
        return current != null;
    }

    public Integer next() {
        Integer item = current.item;
        current = current.next;
        return item;
    }

    public void remove() {
            throw new UnsupportedOperationException();
    }
}
```

Segunda implementação: genérica

API

public class	Bag	
	`Bag()`	cria um saco de itens vazio
`void`	`add(Item item)`	coloca item neste saco
`boolean`	`isEmpty()`	este saco está vazio?
`int`	`size()`	número de Inteitos neste saco
`void`	`startIterator()`	inicializa o iterador
`boolean`	`hasNext()`	há itens a serem iterados?
`Integer`	`next()`	próximo item

Cliente

```
import edu.princeton.cs.algs4.StdOut;

public class Cliente {
    public static void main(String args[]) {
        Bag<Integer> bag = null;
        bag = new Bag<Integer>();
        for (int i=10; i < 20; i++) {
            bag.add(i);
        }
        StdOut.println(bag.size());
        bag.startIterator();
```

```
        while (bag.hasNext()) {
            Integer valor = bag.next();
            StdOut.println("Item " + valor);
        }
        // Bag de Strings
        Bag<String> bagS = new Bag<String>();
        bagS.add("Como");
        bagS.add("é ");
        bagS.add("bom");
        bagS.add("estudar");
        bagS.add("MAC0323!");
        StdOut.println(bagS.size());
        bagS.startIterator();
        while (bagS.hasNext()) {
            StdOut.print(bagS.next() + " ");
        }
        StdOut.println();
    }
}
```

Implementação

```
public class Bag<Item> {

    private Node first;
    private int  n;
    private Node current;

    private class Node {
        private Item item;
        private Node next;
    }

    public Bag () {  // construtor
        first = null;
    }

    public void add(Item item) {
        Node oldfirst = first;
        first = new Node();
        first.item = item;
        first.next = oldfirst;
        n++;
    }

    public int size() {
        return n;
    }
```

```
    public boolean isEmpty() {
        return n == 0;
    }

    public void startIterator() {
        current = first;
    }

    public boolean hasNext(){
        return current != null;
    }

    public Item next() {
        Item item = current.item;
        current = current.next;
        return item;
    }

    public void remove() {
            throw new UnsupportedOperationException();
    }
}
```

Terceira implementação: iterável

API

public class	Bag	implements Iterable
	`Bag()`	cria um saco de Items vazio
`void`	`add(Item item)`	coloca item neste saco
`Iterator<Item>`	`iterator()`	um iterador que percorre os itens do saco
`boolean`	`isEmpty()`	este saco está vazio?
`int`	`size()`	número de Items neste saco

Cliente

```
import edu.princeton.cs.algs4.StdOut;
import java.util.Iterator;

public class Cliente {
    public static void main(String args[]) {
        Bag<Integer> bag = null;
        bag = new Bag<Integer>();
        for (int i=10; i < 20; i++) {
            bag.add(i);
```

```
        }
        StdOut.println(bag.size());
        Iterator<Integer> it = bag.iterator();
        while (it.hasNext()) {
            Integer valor = it.next();
            StdOut.println("Item " + valor);
        }
        // Bag de Strings
        Bag<String> bagS = new Bag<String>();
        bagS.add("Como");
        bagS.add("é ");
        bagS.add("bom");
        bagS.add("estudar");
        bagS.add("MAC0323!");
        StdOut.println(bagS.size());
        for (String s: bagS) {
            StdOut.print(s + " ");
        }
        StdOut.println();
    }
}
```

Implementação

```
// Passo 0:
import java.util.Iterator;

// Passo 1: colocar `implements Iterable<Item>`` na declaração da classe
public class Bag<Item> implements Iterable<Item>{

    private Node first;
    private int  n;
    private Node current;

    private class Node {
        private Item item;
        private Node next;
    }

    public Bag () {  // construtor
        first = null;
    }

    public void add(Item item) {
        Node oldfirst = first;
        first = new Node();
        first.item = item;
        first.next = oldfirst;
        n++;
```

```
        }

        public int size() {
            return n;
        }

        public boolean isEmpty() {
            return n == 0;
        }

        // Passo 2: escrever um método ``Iterator<Item> iterator()``
        public Iterator<Item> iterator() {
            return new BagIterator();
        }

        // Passo 3: escrever a classe que implemanta Iterator
        private class BagIterator implements Iterator<Item> {
            private Node current = first;

            public boolean hasNext(){
                return current != null;
            }

            public Item next() {
                Item item = current.item;
                current = current.next;
                return item;
            }

            public void remove() {
                throw new UnsupportedOperationException();
            }
        }
    }
```

Range

Aqui vai outro exemplo simples de classe iterável. Esse exemplo foi copiado de [https://codereview.stackexchange.com/example-of-an-iterable-and-an-iterator-in-java)

```
import java.util.NoSuchElementException;
import java.util.Iterator;
import edu.princeton.cs.algs4.StdOut;

public class Range implements Iterable<Integer> {
    private final int start;
    private final int   end;

    public Range(int start, int end) {
```

```
        this.start = start;
        this.end = end;
    }

    public Iterator<Integer> iterator() {
        return new RangeIterator(start, end);
    }

    // Inner class example
    private static class RangeIterator implements
                    Iterator<Integer> {
        private int current;
        private final int end;

        public RangeIterator(int start, int end) {
            this.current = start;
            this.end     = end;
        }

        public boolean hasNext() {
            return current < end;
        }

        public Integer next() {
            if (!hasNext()) {
                throw new NoSuchElementException();
            }
            int val = current;
            current ++;
            return val;
        }

        public void remove() {
            throw new UnsupportedOperationException();
        }
    }

    private static void help() {
        String s = "Uso: java Range ini fim\n"
            + "    ini= inteiro início do intervalo\n"
            + "    fim= inteiro fim do intervalo";
        StdOut.println(s);
    }

    public static void main(String[] args) {

        if (args.length != 2) {
            help();
            return;
        }
```

```
        int ini;
        int fim;
        try {
            ini = Integer.parseInt(args[0]);
            fim = Integer.parseInt(args[1]);
        } catch (NumberFormatException e) {
            StdOut.println("NumberFormatException: " + e.getMessage());
            help();
            return;
        }
        Range range = new Range(ini, fim);

        // Long way
        StdOut.println("Iterando baixo nível");
        Iterator<Integer> it = range.iterator();
        while(it.hasNext()) {
            int cur = it.next();
            System.out.println(cur);
        }

        // Shorter, nicer way:
        // Read ":" as "in"
        StdOut.println("Iterando alto nível");
        for(Integer cur : range) {
            System.out.println(cur);
        }
    }
}
```

Receita para construir uma classe iterável

Leia Bags, Queues, and Stacks.

Passo 0

Incluir o seguinte `import` para que possamos no referir a interface `java.util.Iterator`:

```
import java.util.Iterator;
```

Passo 1

Adicionar `implements Iterable<Item>` no final da declaração da classe. Isso indica que o objeto que definimos será iterável e nos comprometemos a especificar o método `iterator()`, como especficado na interface `java.lang.Iterable`

```
   public interface Iterable<Item> {
       public Iterator<Item> iterator();
```

```
}
```

Exemplo:

```
public class Bag<Item> implements Iterable<Item> {
    ...
}
```

Passo 2

Implementar um método `iterator()` como prometido. Esse método retorna um objeto da classe que implementa a interface `Iterator`

```
public interface Iterator<Item> {
    boolean hasNext();
    Item next();
    void remove();
}
```

Por exemplo:

```
public Iterator<Item> iterator() {
    return new BagIterator();
}
```

Passo 3

Implemente a subclasse que implementa a interface `Iterator` incluindo od método `hasNext()`, `next()` eremove()`Usamos sempre o método vazio para o opcional método`remove()‘ pois intercalar iteração com uma operação que modifica a estruturas de dados é melhor ser evitada.

Exemplo:

```
private class BagIterator implements Iterator<Item> {
    private Node current = first;

    public boolean hasNext(){
        return current != null;
    }

    public Item next() {
        if (!hasNext()) {
            throw new NoSuchElementException();
        }
        Item item = current.item;
        current = current.next;
        return item;
    }

    public void remove() {
        throw new UnsupportedOperationException();
```

```
    }
}
```