Técnicas de Programação Avançada

TCC-00.174

Prof.: Anselmo Montenegro

www.ic.uff.br/~anselmo

anselmo@ic.uff.br

Conteúdo: Coleções

Material elaborado com contribuição do Professor Luiz André

• Uma coleção, as vezes chamada *container* é um objeto que agrupa múltiplos elementos de natureza comum em único objeto

•Coleções são usadas para armazenar, recuperar, manipular efetuar a comunicação de dados agregados

•Exemplos de coleções:

- Mão de um jogo de Poker
- Uma pasta de emails
- Agenda telefônica

• Framework de coleções: é um arquitetura unificada para representação e manipulação de coleções

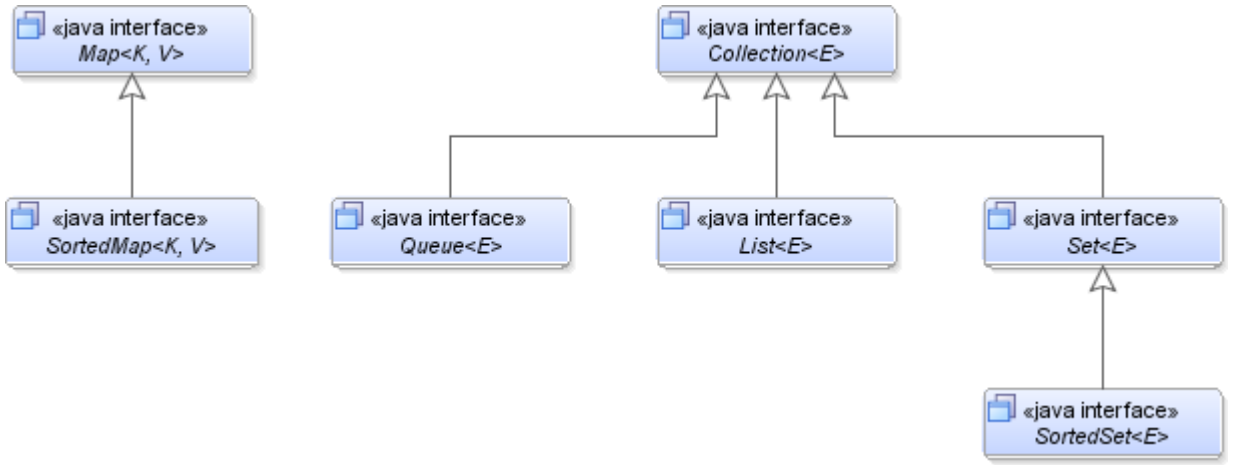
•É composta de:

- Interfaces: tipos abstratos que representam as coleções
- Implementações: implementações concretas das coleções
- Algoritmos: métodos que executam computações uteis tais como busca, ordenação em objetos que implementam as interfaces

- **Benefício do Framework de Coleções de Java:**
  - **Reduzir o esforço de programação**
  - **Aumentar a velocidade e qualidade do programa**
  - **Permitir a interoperabilidade entre APIs não relacionadas**
  - **Reduzir o esforço para aprender e usar novas APIs**
  - **Reduzir o esforço de projeto de novas APIs**
  - **Viabiliza reuso de software**

Coleções - são estruturas de dados para armazenamento de objetos

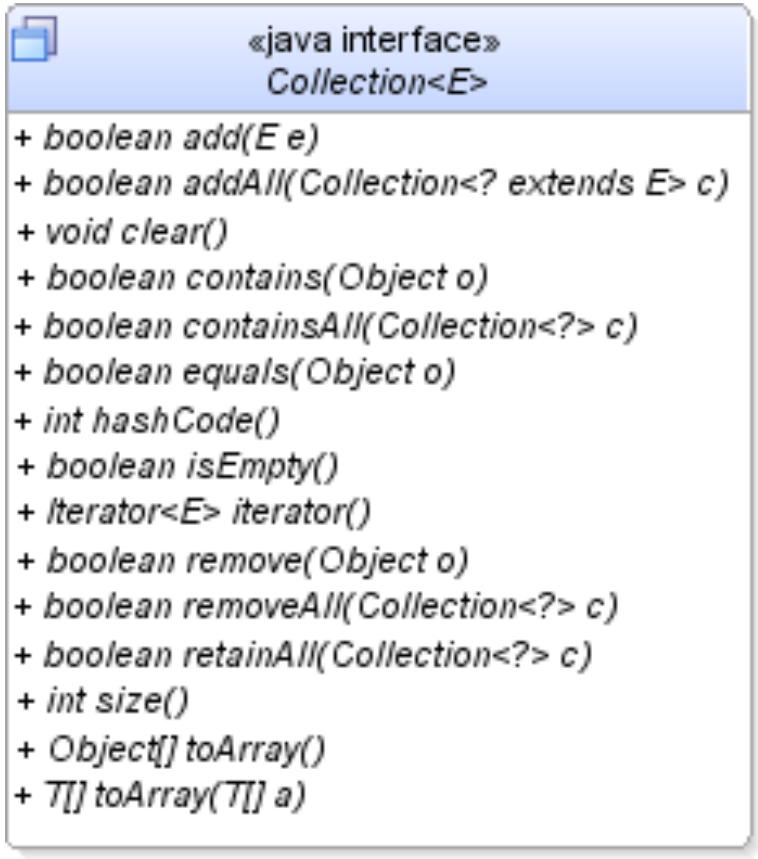
Tipos de coleções em Java



Collection

Nível mais alto de abstração

Contém as operações comuns a todas as coleções



• Como percorrer coleções:

•For each:

```
for (Object o : collection)
        System.out.println(o);
```

•Iterator:

```
static void filter(Collection<?> c) {
        for (Iterator<?> it = c.iterator(); it.hasNext(); )
                if (!cond(it.next())) it.remove();
 }
```

- *Bulk operations:* **executa operações em uma coleção completa**
  - **containsAll — retorna verdadeiro se a coleção alvo contêm todos os elementos na coleção especificada**
  - **addAll — adiciona todos os elementos da coleção especificada na coleção alvo**
  - **removeAll — remove da coleção alvo todos os elementos que também estão contidos na coleção especificada**
  - **retainAll — remove da coleção alvo todos os elementos que não estão contidos na coleção especificada**
  - **clear — remove todos os elementos da coleção**

- **Operações de conversão de coleção para array:**

```
Object[] a = c.toArray();

String[] a = c.toArray(new String[0]);
```

- **Operações de conversão de coleção para array:**

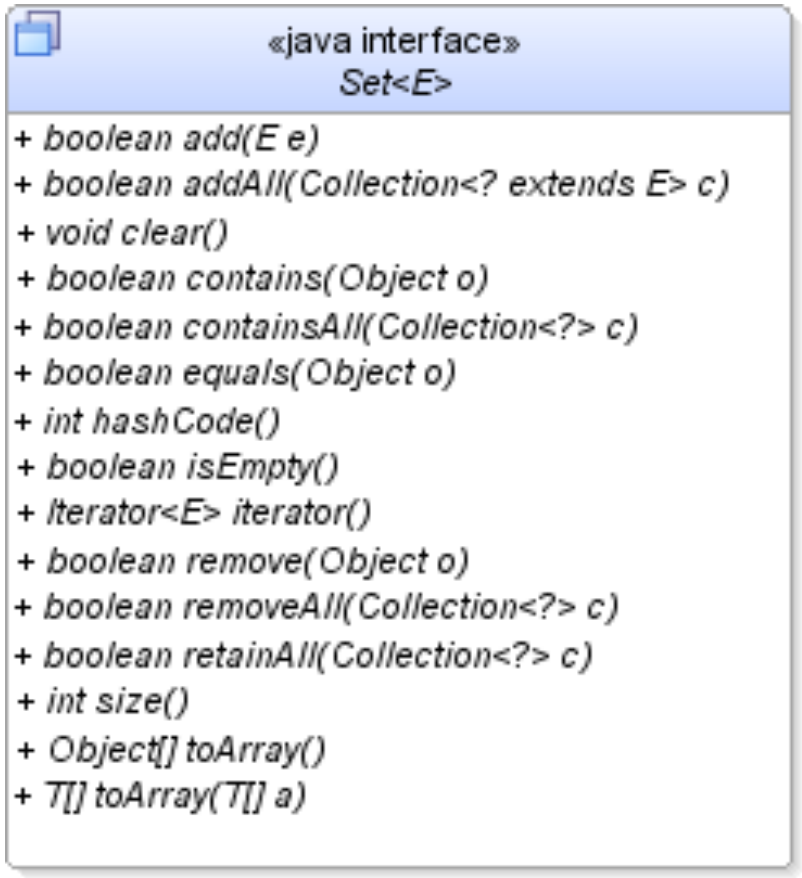
```
Object[] a = c.toArray();

String[] a = c.toArray(new String[0]);
```

Set

Conjunto de objetos que não pode conter elementos duplicados

Equivale ao conceito matemático de conjunto



- Exemplo: remover duplicatas

```
public class FindDups { public static void main(String[] args) {
  Set<String> s = new HashSet<String>();
  for (String a : args)
     s.add(a);
     System.out.println(s.size() + " distinct words: " + s;

}
```

- **Exemplo: união, interseção e diferença**

```
Set<Type> union = new HashSet<Type>(s1);
union.addAll(s2);

Set<Type> intersection = new HashSet<Type>(s1);
intersection.retainAll(s2);

Set<Type> difference = new HashSet<Type>(s1);
difference.removeAll(s2);
```

- **Exemplo: Lista de únicas e duplicatas**

```
public class FindDups2 { public static void main(String[] args) {
  Set<String> uniques = new HashSet<String>();
  Set<String> dups = new HashSet<String>();
  for (String a : args){
    if (!uniques.add(a))
      dups.add(a); // Destructive set-difference uniques.removeAll
   (dups);
    System.out.println("Unique words: " + uniques); System.out.println
    ("Duplicate words: " + dups);
  }
}
```

- **Exemplo: Lista de únicas e duplicatas**

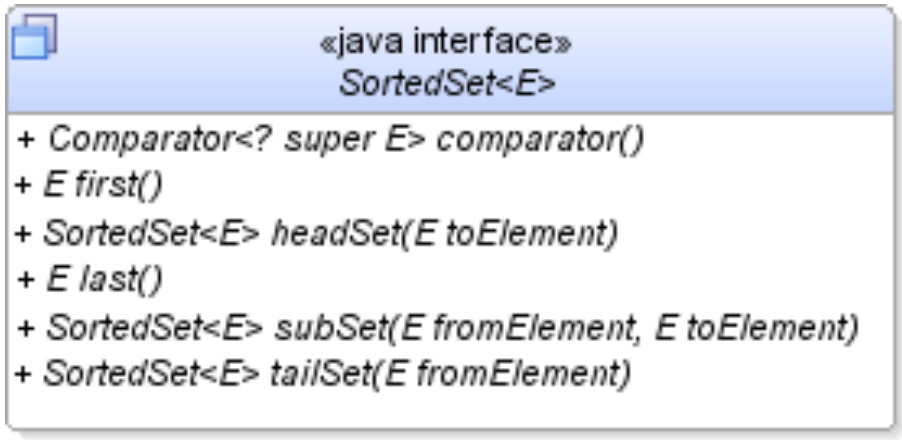
```
public class FindDups2 { public static void main(String[] args) {
  Set<String> uniques = new HashSet<String>();
  Set<String> dups = new HashSet<String>();
  for (String a : args){
    if (!uniques.add(a))
      dups.add(a); // Destructive set-difference uniques.removeAll
   (dups);
    System.out.println("Unique words: " + uniques); System.out.println
    ("Duplicate words: " + dups);
  }
}
```

- **Exemplo: diferença simétrica**

```
Set<Type> symmetricDiff = new HashSet<Type>(s1);
symmetricDiff.addAll(s2);
Set<Type> tmp = new HashSet<Type>(s1); tmp.retainAll(s2);
symmetricDiff.removeAll(tmp);
```

SortedSet

Conjunto objetos classificados em ordem crescente



- **Exemplo: diferença simétrica**

```
Set<Type> symmetricDiff = new HashSet<Type>(s1);
symmetricDiff.addAll(s2);
Set<Type> tmp = new HashSet<Type>(s1); tmp.retainAll(s2);
symmetricDiff.removeAll(tmp);
```

List

Conjunto ordenado de objetos

Pode conter elementos duplicados

- Exemplo: swap e shuffle

```
public static <E> void swap(List<E> a, int i, int j) {
  E tmp = a.get(i);
  a.set(i, a.get(j));
  a.set(j, tmp);
 }

public static void shuffle(List<?> list, Random rnd) {
  for (int i = list.size(); i > 1; i--){
    swap(list, i - 1, rnd.nextInt(i));
 }
```

- **Exemplo: shuffle da classe Collections**

```
public class Shuffle { public static void main(String[] args) {
   List<String> list = new ArrayList<String>();
   for (String a : args){
      list.add(a);
      Collections.shuffle(list, new Random());
      System.out.println(list);
   }
}
```

- Exemplo: transformando um array em uma lista

```
public class Shuffle { public static void main(String[] args) {
    List<String> list = Arrays.asList(args);
    Collections.shuffle(list);
    System.out.println(list);
  }
}
```

• Exemplo: percorrimento de uma lista

•ListIterator – herda de Iterator; permite varrer uma lista em ambas as direções fornencendo as operações next(), hasNext(), previous e hasPrevious()

```
for (ListIterator<Type> it = list.listIterator(list.size()); it.hasPrevious(); ) {
    Type t = it.previous();
    ...
}
```

• A interface List tem duas formas do ListIterator: uma com argumento e a outra sem argumento

```
ListIterator<E> listIterator()
ListIterator<E> listIterator(int index)
```

•A forma sem argumento retorna o iterador no inicio da lista

•A forma com argumento inteiro retorna um ListIterator posicionado no no index especificado

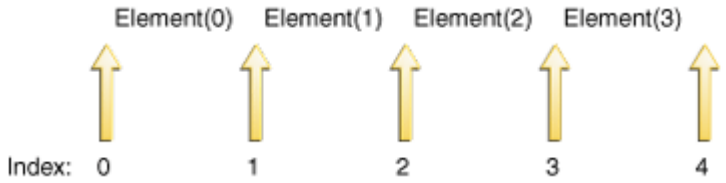
•O índice corresponde ao elemento que será retornado ao invocar-se next()

•Uma chamada a previous retornará index-1

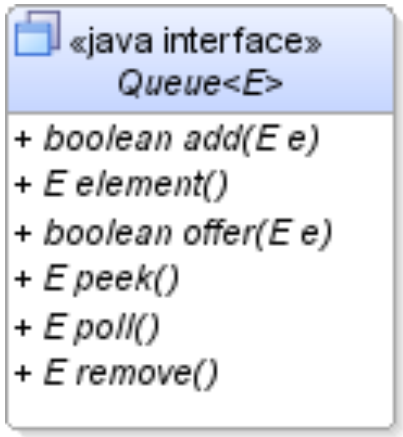
• Em uma lista com n elementos, existem n+1 valores válidos para índice de 0 a n inclusive

• O cursor está sempre entre 2 elementos: um que seria retornado pela chamada a previous() e aquela que seria chamada por next()

• Os n+1 valores válidos correspondem aos n+1 espaços entre os elementos



`Queue`



Fila de objetos

Ordenação pode ser FIFO ou determinada por prioridade

Exemplo: simulação de um heap sort usando uma PriorityQueue

```
static <E> List<E> heapSort(Collection<E> c) {
  Queue<E> queue = new PriorityQueue<E>(c);
  List<E> result = new ArrayList<E>();

  while (!queue.isEmpty())
    result.add(queue.remove());
  return result;
}
```

Map

Conjunto de objetos acessados por chaves

Não pode conter chave duplicadas

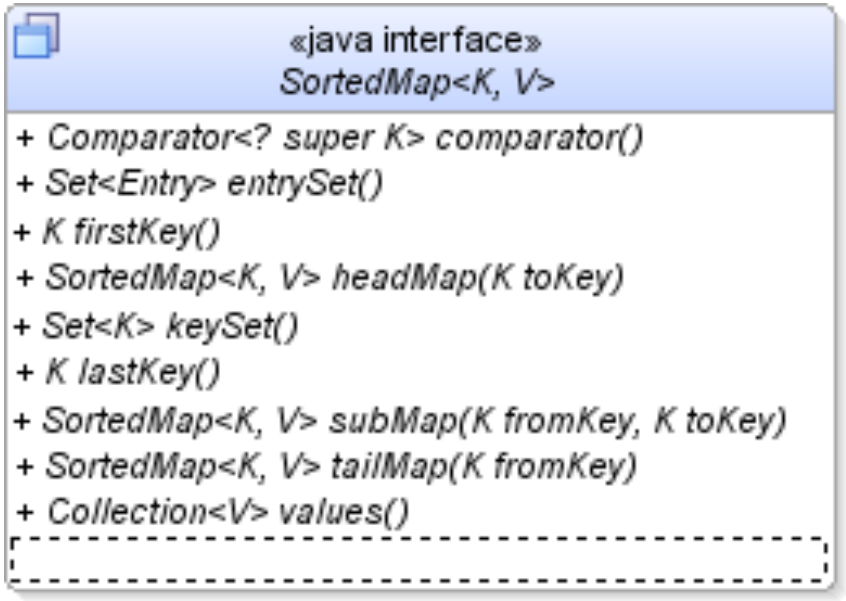
Cada chave está associada a somente um objeto

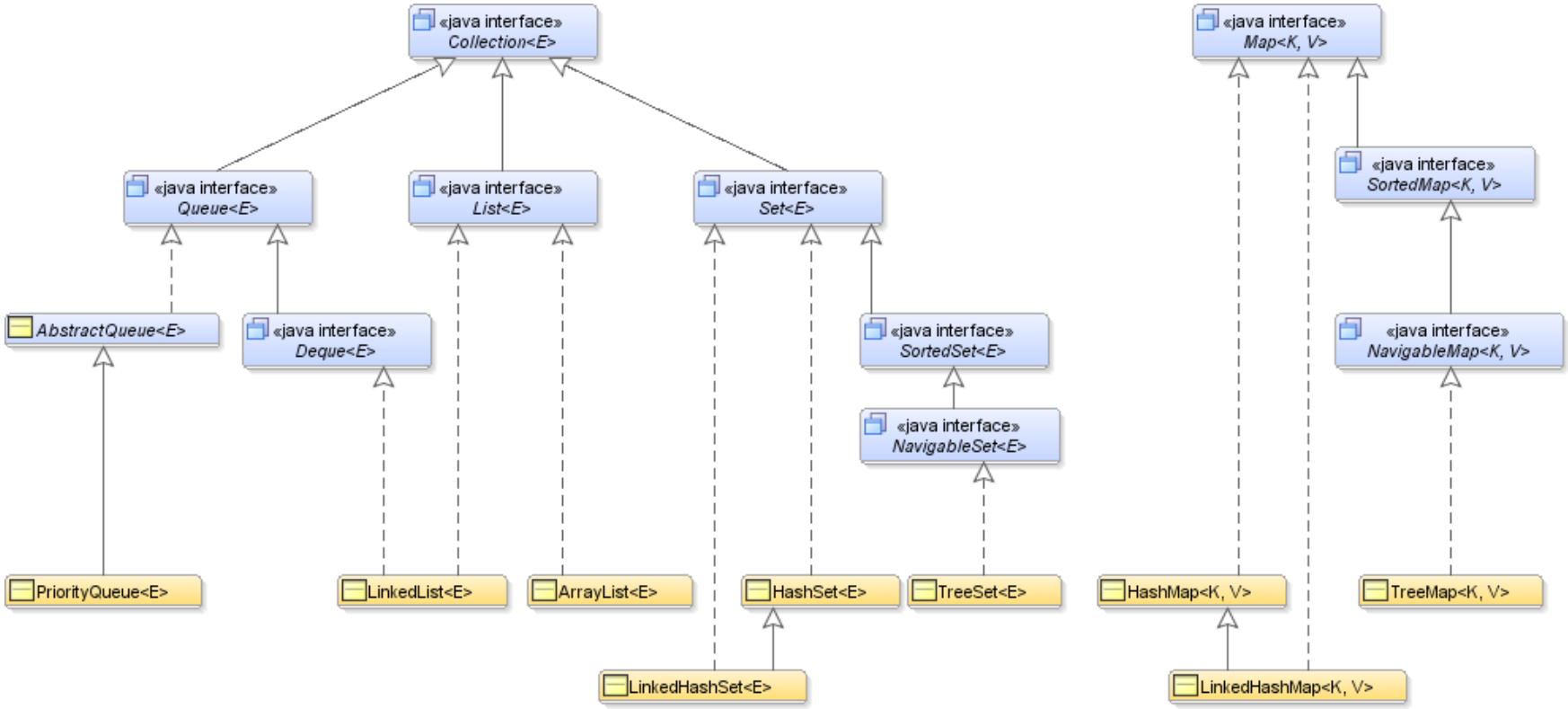
Exemplo: histograma de frequencias

```
public class Freq {
  public static void main(String[] args) {
     Map<String, Integer> m = new HashMap<String, Integer>();
    // Initialize frequency table from command line
    for (String a : args) {
      Integer freq = m.get(a);
      m.put(a, (freq == null) ? 1 : freq + 1);
    }
    System.out.println(m.size() + " distinct words:");
    System.out.println(m);
  }
}
```

SortedMap

Mantém as chaves classificadas em ordem crescente




«java interface»
Collection<E>
«java interface»
Queue<E>
«java interface»
List<E>
«java interface»
Set<E>
AbstractQueue<E>
«java interface»
Deque<E>
«java interface»
SortedSet<E>
«java interface»
NavigableSet<E>
PriorityQueue<E>
LinkedList<E>
ArrayList<E>
HashSet<E>
TreeSet<E>
LinkedHashSet<E>
«java interface»
Map<K, V>
«java interface»
SortedMap<K, V>
«java interface»
NavigableMap<K, V>
HashMap<K, V>
TreeMap<K, V>
LinkedHashMap<K, V>

•Como comparar objetos considerando uma ordem diferente da ordem natural do tipo considerado?

•Como viabilizar a ordenação de objetos que não implementam a interface Comparable?

•Solução: usar a interface Comparator

```
public interface Comparator<T> {
  int compare(T o1, T o2);
 }
```

•Como comparar objetos considerando uma ordem diferente da ordem natural do tipo considerado?

•Como viabilizar a ordenação de objetos que não implementam a interface Comparable?

•Solução: usar a interface Comparator

```
public interface Comparator<T> {
  int compare(T o1, T o2);
 }
```

- Exemplo: considere uma classe Empregado com ordenação natural dada por nome

- Como ordená-la por um critério baseando em outro campo?

```
public class Employee implements Comparable<Employee> {
    public Name name() { ... };
    public int number() { ... };
    public Date hireDate() { ... };
    ...
}
```

- Exemplo: considere uma classe Empregado com ordenação natural dada por nome

- Como ordená-la por um critério baseando em outro campo?

```
public class Employee implements Comparable<Employee> {
    public Name name() { ... };
    public int number() { ... };
    public Date hireDate() { ... };
    ...
}
```

```
public class EmpSort {
   static final Comparator<Employee> SENIORITY_ORDER = new
    Comparator<Employee>() {

        public int compare(Employee e1, Employee e2) {
            return    e2.hireDate().compareTo(e1.hireDate());
      }
   };

  // Employee database static final Collection<Employee> employees = ... ;
   public static void main(String[] args) {
        List<Employee> e = new ArrayList<Employee>(employees);
        Collections.sort(e, SENIORITY_ORDER);
        System.out.println(e);
   }
}
```

Orientação a Objetos

Coleções - Implementações

- `HashSet`
  - Ordena objetos em hash table
- `TreeSet`
  - Ordena objetos em árvore
  - Objetos armazenados devem implementar interface `Comparable`
- `LinkedHashSet`
  - Armazena a ordem de entrada dos elementos em uma lista duplamente encadeada
  - Ordem de iteração preserva ordem de entrada
- `ArrayList`
  - Armazena objetos em lista simples
- `LinkedList`
  - Lista duplamente encadeada
  - FIFO
- `PriorityQueue`
  - Lista

- `TreeMap`
  - Ordena chave em árvore
- `HashMap`
  - Ordena chave em hash table
- `LinkedHashMap`
  - Mantém a ordem em que os elementos foram incluídos no mapa em uma lista duplamente encadeada
  - A ordem de inserção é preservada em iterações sobre as chaves

- Exemplos
  - Dias da semana: segunda, terça, etc.
  - Frutas: morango, mamão, abacaxi, etc.
- A ordem dos valores é definida
- Os elementos são literais constantes
- Definição Java

```
public enum Day {
    SEGUNDA, TERCA, QUARTA, QUINTA, SEXTA, SABADO, DOMINGO;
}
```

- Comparação

```
Day day = Day.SEGUNDA;          System.out.println
(day.compareTo(Day.TERCA));
```

```
public enum Planet {
  MERCURY (3.303e+23, 2.4397e6),
  VENUS (4.869e+24, 6.0518e6),
  EARTH (5.976e+24, 6.37814e6),
  MARS (6.421e+23, 3.3972e6),
  JUPITER (1.9e+27, 7.1492e7),
  SATURN (5.688e+26, 6.0268e7),
  URANUS (8.686e+25, 2.5559e7),
  NEPTUNE (1.024e+26, 2.4746e7);

  private final double mass;
  private final double radius;
  public static final double G = 6.67300E-11;

...
```

```
...
  Planet(double mass, double radius) {
    this.mass = mass; this.radius =
    radius;
  }
  private double mass() {
    return mass;
  }
  private double radius() {
    return radius;
  }
  public double surfaceGravity() {
    return G * mass / (radius *
    radius);
  }
  public double surfaceWeight(double
  otherMass){
    return otherMass *
    surfaceGravity();
  }
}
```

```
public static void main(String[] args) {
   Planet planet = Planet.MARS;
   System.out.println(planet.surfaceWeight(72));
}
```

- Coleção de valores enumeráveis
  - **class EnumSet<E extends Enum<E>>**
  - **public static <E extends Enum<E>> EnumSet<E> noneOf**
  - **(Class<E> elementType)**

```
Set<Day> myEnum = EnumSet.allOf
(Day.class);

for (Day myday : myEnum)
    System.out.println(myday.toString());
```

- Collections: http://docs.oracle.com/javase/tutorial/collections/
- Collections Interfaces : http://docs.oracle.com/javase/tutorial/collections/interfaces/index.html
- Collections Implementations : http://docs.oracle.com/javase/tutorial/collections/implementations/index.html
- Collections Algorithms: http://docs.oracle.com/javase/tutorial/collections/algorithms/index.html