Colecções Java

- O Java oferece um conjunto de classes que implementam as estruturas de dados mais utilizadas
  - oferecem uma API consistente entre si
  - permitem que sejam utilizadas com qualquer tipo de objecto - são parametrizadas por tipo

- Poderemos representar:
  - ArrayList<Aluno> alunos
  - HashSet<Aluno> alunos;
  - HashMap<String, Aluno> turmaAlunos;
  - TreeMap<String, Docente> docentes;
  - Stack<Pedido> pedidosTransferência;

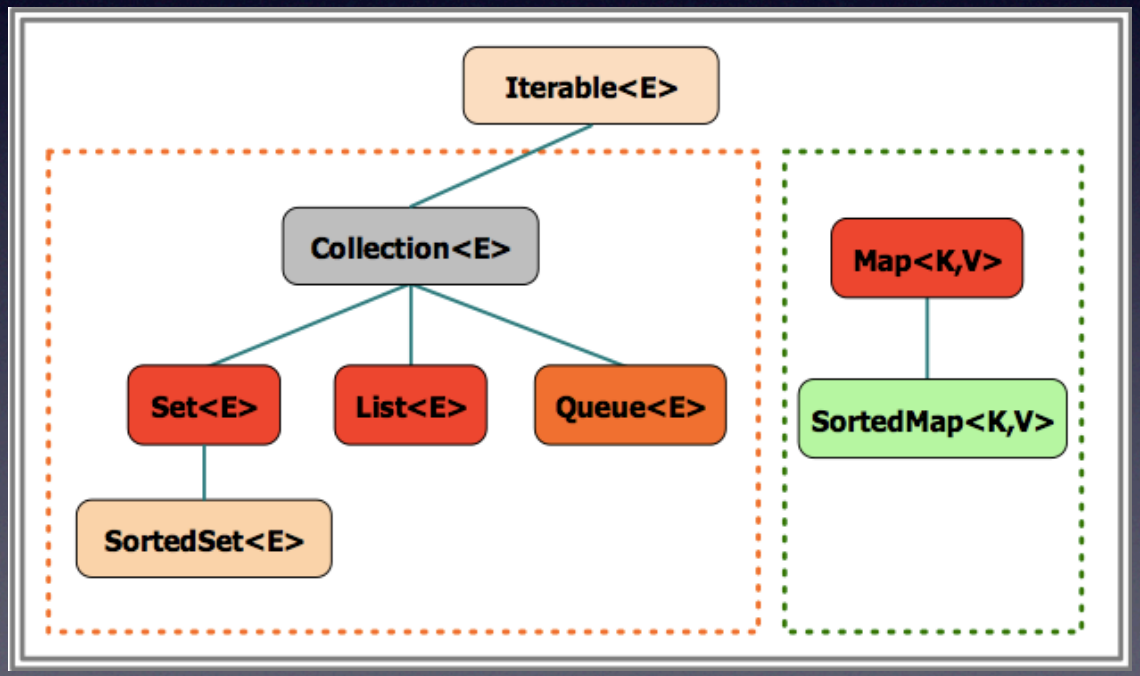
- Ao fazer-se ArrayList<Circulo> passa a ser o compilador a testar, e validar, que só são utilizados objectos do tipo Circulo no arraylist.
  - isto dá uma segurança adicional aos programas, pois em tempo de execução não teremos erros de compatibilidade de tipos
  - os tipos de dados são verificados em tempo de compilação

JCF

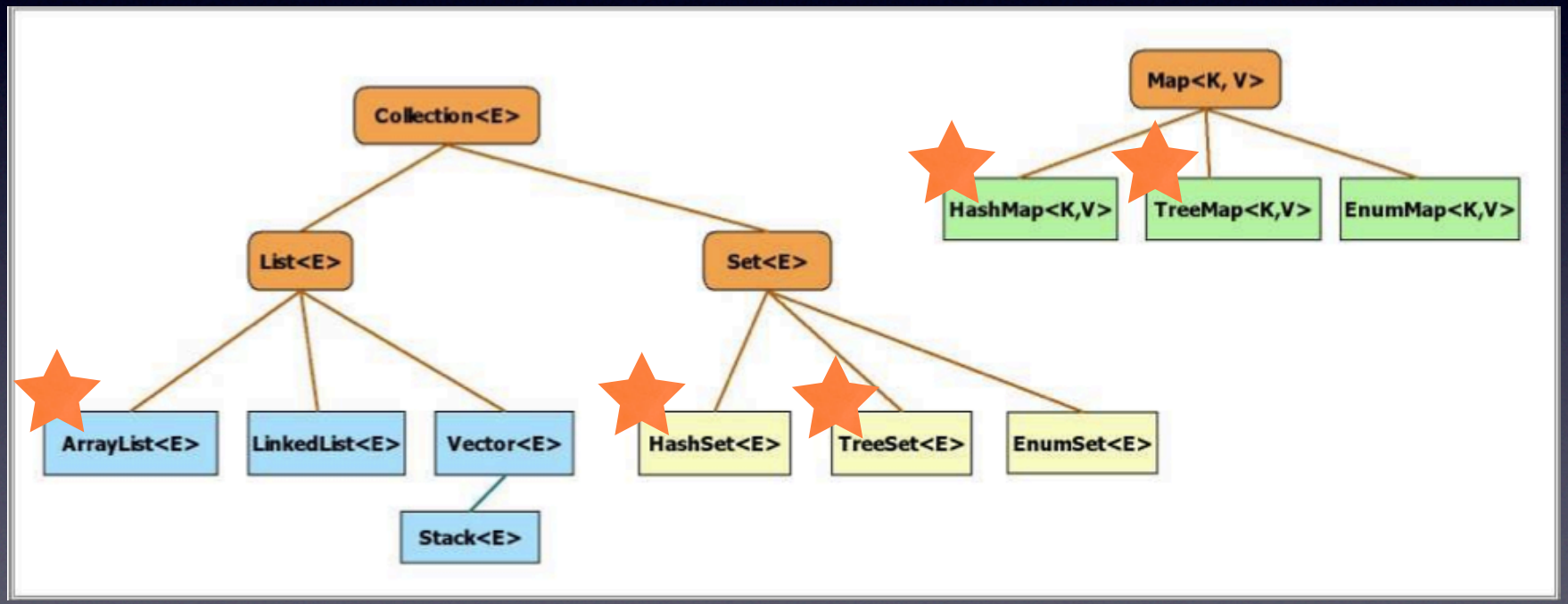
- O JCF (Java Collections Framework) agrupa as várias classes genéricas que correspondem às implementações de referência de:
  - Listas: API de **List<E>**
  - Conjuntos: API de **Set<E>**
  - Correspondências unívocas: API de **Map<K,V>**

Estrutura da JCF

- Existe uma arrumação por API (interfaces)



- Para cada API (interface) existem diversas implementações (a escolher consoante critérios do programador)



ArrayList<E>

- As classes da Java Collections Framework são exemplos muito interessantes de codificação
- Como o código destas classes está escrito em Java, é possível ao programador observar como é que foram implementadas

ArrayList<E>: v.i. e construtores

```
public class ArrayList<E> extends AbstractList<E>
        implements List<E>, RandomAccess, Cloneable, java.io.Serializable
{
    private static final long serialVersionUID = 8683452581122892189L;

    /**
     * The array buffer into which the elements of the ArrayList are stored.
     * The capacity of the ArrayList is the length of this array buffer.
     */
    private transient Object[] elementData;

    /**
     * The size of the ArrayList (the number of elements it contains).
     *
     * @serial
     */
    private int size;

    /**
     * Constructs an empty list with the specified initial capacity.
     *
     * @param  initialCapacity  the initial capacity of the list
     * @throws IllegalArgumentException if the specified initial capacity
     *         is negative
     */
    public ArrayList(int initialCapacity) {
        ...
        this.elementData = new Object[initialCapacity];
    }

    /**
     * Constructs an empty list with an initial capacity of ten.
     */
    public ArrayList() {
        this(10);
    }
```

ArrayList<E>: existe?

```
public boolean contains(Object o) {
    return indexOf(o) >= 0;
}

public int indexOf(Object o) {
    if (o == null) {
        for (int i = 0; i < size; i++)
            if (elementData[i]==null)
                return i;
    } else {
        for (int i = 0; i < size; i++)
            if (o.equals(elementData[i]))
                return i;
    }
    return -1;
}
```

ArrayList<E>: inserir

```
public boolean add(E e) {
    ensureCapacityInternal(size + 1);  // Increments modCount!!
    elementData[size++] = e;
    return true;
}


public void add(int index, E element) {
    rangeCheckForAdd(index);

    ensureCapacityInternal(size + 1);  // Increments modCount!!
    System.arraycopy(elementData, index, elementData, index + 1,
                     size - index);
    elementData[index] = element;
    size++;
}
```

ArrayList<E>: get e set

```
public E get(int index) {
    rangeCheck(index);

    return elementData(index);
}


public E set(int index, E element) {
    rangeCheck(index);

    E oldValue = elementData(index);
    elementData[index] = element;
    return oldValue;
}
```

Colecções Java

- Tipos de colecções disponíveis:
  - listas (definição em List<E>)
  - conjuntos (definição em Set<E>)
  - queues (definição em Queue<E>)
- noção de família (muito evidente) nas APIs de cada um destes tipos de colecções.

List<E>

- Utilizar sempre que precise de manter ordem
- O método add não testa se o objecto existe na colecção
- O método contains verifica sempre o resultado de equals
- Implementação utilizada: **ArrayList<E>**

ArrayList<E>

Categoria de Métodos	API de ArrayList<E>

Construtores	`new ArrayList<E>()`
	`new ArrayList<E>(int dim)`
	`new ArrayList<E>(`*`Collection`*`)`

Inserção de elementos	`add(E o); add(int index, E o);` `addAll(`*`Collection`*`); addAll(int i, `*`Collection`*`);`
Remoção de elementos	`remove(Object o); remove(int index);` `removeAll(`*`Collection`*`); retainAll(`*`Collection`*`)`

Consulta e comparação de conteúdos	`E get(int index); int indexOf(Object o);`
	`int lastIndexOf(Object o);`
	`boolean contains(Object o); boolean isEmpty();`
	`boolean containsAll(`*`Collection`*`); int size();`

Criação de Iteradores	`Iterator<E> iterator();`
	`ListIterator<E> listIterator();`
	`ListIterator<E> listIterator(int index);`

Modificação	`set(int index, E elem); clear();`
Subgrupo	`List<E> sublist(int de, int ate);`
Conversão	`Object[] toArray();`
Outros	`boolean equals(Object o); boolean isEmpty();`

```java
import java.util.ArrayList;
public class TesteArrayList {
  public static void main(String[] args) {

    Circulo c1 = new Circulo(2,4,4.5);
    Circulo c2 = new Circulo(1,4,1.5);
    Circulo c3 = new Circulo(2,7,2.0);
    Circulo c4 = new Circulo(3,3,2.0);
    Circulo c5 = new Circulo(2,6,7.5);

    ArrayList<Circulo> circs = new ArrayList<Circulo>();
    circs.add(c1.clone());
    circs.add(c2.clone());
    circs.add(c3.clone());

    System.out.println("Num elementos = " + circs.size());
    System.out.println("Posição do c2 = " + circs.indexOf(c2));

    for(Circulo c: circs)
      System.out.println(c.toString());

  }
}
```

Set<E>

- Utilizar sempre que se quer garantir ausência de elementos repetidos
- O método add testa se o objecto existe
- O método contains utiliza a lógica do equals, mas não só...
- Duas implementações: **HashSet<E>** e **TreeSet<E>**

Set<E>

Categoria de Métodos	API de Set<E>
Inserção de elementos	`add(E o);` `addAll(Collection); addAll(int i, Collection);`
Remoção de elementos	`remove(Object o); remove(int index);` `removeAll(Collection); retainAll(Collection)`
Consulta e comparação de conteúdos	`boolean contains(Object o); boolean isEmpty();` `boolean containsAll(Collection); int size();`
Criação de Iteradores	`Iterator<E> iterator();`
Modificação	`clear();`
Subgrupo	`List<E> sublist(int de, int ate);`
Conversão	`Object[] toArray();`
Outros	`boolean equals(Object o); boolean isEmpty();`

HashSet<E>

- O método add calcula o valor do hash de E para determinar a posição do objecto na estrutura de dados
- O método contains necessita de saber o hash do objecto para determinar a posição em que o encontra
- Logo, não chega ter o equals definido
  - é necessário ter o método **hashcode()**

TreeSet<E>

- O método add determina a posição na árvore em que deve colocar o objecto
- É necessário fornecer um método de comparação dos objectos - **compare()** ou **compareTo()**
  - sem este método de comparação não é possível utilizar o TreeSet, a não ser para tipos de dados simples (String, Integer, etc.)