O método toString

- a informação deve ser concisa (sem *acuçar de ecran*), mas ilustrativa
- todas as classes devem implementar este método
- caso não seja implementado a resposta será:

getClass().getName() + '@' + Integer.toHexString(hashCode())

O método toString

- implementação *normal* de toString na classe Aluno

```
/**
 * Implementação do método toString
 * comum na maioria das classes Java
 *
 * @return     uma string com a informação textual do objecto aluno
 */
public String toString() {

    return("Numero:" + this.numero + "Nome:"+ this.nome + "Nota:" + this.nota);

}
```

- o operador "+" é a concatenação de Strings, sempre que o resultado seja uma String

- Strings são objectos imutáveis, logo não crescem, o que as torna muito ineficientes
  - o mesmo método, de forma mais eficiente, na medida em que as concatenações de Strings são muito pesadas

```
/**
  * Implementação do método toString
  * comum na maioria das classes Java
  *
  * @return     uma string com a informação textual do objecto aluno
  */
 public String toString() {
     StringBuilder sb= new StringBuilder();

     sb.append("Numero: ");
     sb.append(this.numero+"\n");
     sb.append("Nome: ");
     sb.append(this.nome+"\n");
     sb.append("Nota: ");
     sb.append(this.nota+"\n");

     return sb.toString();
 }
```

...completar a classe Turma

- equals

```
/**
 * Método equals.
 * Utiliza o método privado getLstAlunos para efectuar a comparação entre
 * duas instância de turma.
 */

public boolean equals(Turma umaTurma) {
    if (umaTurma != null)
      return (this.designacao.equals(umaTurma.getDesignacao())

              && this.ocupacao == umaTurma.getOcupacao()
              && Arrays.equals(this.lstalunos,umaTurma.getLstAlunos()));
    else
      return false;
}
```

  - nesta versão recorreu-se ao método equals da classe Arrays
  - é necessário garantir que a remoção de alunos não deixa "lixo" no *array* lstalunos

- toString

```
/**
 * Método toString por questões de compatibilização com as restantes
 * classes do Java.
 *
 * Como o toString é estrutural e a classe Aluno tem esse método
 * implementado o resultado é o esperado.
 *
 */
public String toString() {
    StringBuilder sb = new StringBuilder();

    sb.append("Designação: "); sb.append(this.designacao+"\n");

    sb.append("Alunos: "+"\n"); sb.append(this.lstalunos.toString());

    return sb.toString();
}
```

- clone

```
public Turma clone() {
  return new Turma(this);
}
```

Variáveis e métodos de classe

- as classes não deixam de ser objectos
  - objectos que guardam o que é comum a todas as instâncias
  - **apenas um** objecto-classe por classe
- se objectos possuem estado e comportamento, então podemos extrapolar e dizer que a classe também tem:
  - variáveis e métodos de classe

- os métodos de classe são activados a partir de mensagens que são enviadas para a classe.
  - exemplo: Ponto2D.metodo()
- se uma classe possui variáveis de classe o acesso a essas variáveis deverá ser feito através dos métodos de classe
  - métodos de instância => var. instância
  - métodos de classe => var. classe

- o que é que se pode guardar como variável de classe?
  - valores que sejam comuns a todas os objectos instância
  - não faz sentido colocar estes valores em todos os objectos (repetição)

- Imagine-se que numa classe Conta Bancária se pretende:

  a) saber quantas contas foram criadas (possível saber quantas existem?)

  b) definir uma taxa de juro comum a todas as contas

- como é que poderemos satisfazer os requisitos expressos em a) e b)?

- uma variável que guarde o número de contas criadas não é certamente uma variável de instância
    - actualização do contador em todas as instâncias?
    - redundância?
- teriam de se ter implementados mecanismos de comunicação entre todas as instâncias!!

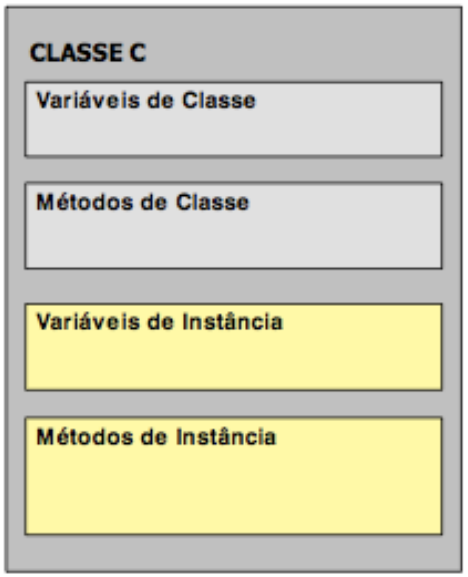
- as variáveis de classe servem para guardar *informação global* a todas as instâncias
- podem também ser utilizadas para guardar constantes que são utilizadas pelos diversos objectos instância
  - exemplo: Math.PI

- os métodos de classe fazem o acesso às variáveis de classe
- aos métodos de classe aplicam-se as mesmas regras de visibilidade que se aplicam aos métodos de instância
- os métodos de classe são sempre acessíveis às instâncias, mas métodos de classe **não** tem acesso aos métodos de instância

- como se declaram métodos e variáveis de classe?
    - utilizando o prefixo **static**
- a definição de classe passa a ter:
    - declaração de variáveis de classe
    - declaração de métodos de classe
    - declaração de varíaveis de instância
    - declaração de métodos de instância

Estrutura de uma classe

- estrutura tipo de uma classe



Classe PMMB

- seja uma classe PMMB (porta moedas multibanco)
  - queremos acrescentar informação sobre o número total de PMMB's criados e sobre o valor total de saldo existente em todos os cartões

- duas variáveis de classe e respectivos métodos de acesso

```
public class PMMB {
   // Variáveis de Classe
   public static int numPMMB = 0;
   public static double saldoTotal = 0.0;

   // Métodos de Classe
   public static int getNumPMMB() {
      return numPMMB;
   }
   public static double getSaldoTotal() {
      return saldoTotal;
   }
   public static void incNumPMMB() {
      numPMMB++;
   }
   public static void actSaldoTotal(double valor) {
      saldoTotal += valor;
   }
```

```
// Variáveis de Instância
private String codigo;
private String titular;
private double saldo;
private int numMovs;  // total de movimentos
```

- onde é que se actualiza a informação do número de cartões criados?
  - no construtor de PMMB

```
public PMMB() {
   codigo = ""; titular = "";
   saldo = 0.0; numMovs = 0;
   PMMB.actSaldoTotal(0);
   PMMB.incNumPMMB();
}
```

```
// Métodos de Instância
. . . .
public void carregaPM(double valor) {
   saldo = saldo + valor;
   numMovs++; actSaldoTotal(valor);
}
// pré-condição
public boolean prePaga(double valor) {
   return saldo >= valor ;
}

public void pagamento(double valor) {
   saldo = saldo – valor;
   numMovs++; actSaldoTotal(-valor);
}
```

- para não confundir quem lê, as invocações anteriores poderiam ter sido feitas na forma:
  - PMMB.actSaldoValorTotal(valor)

- os métodos de classe anteriormente definidos devem ser utilizados como:

```
int pMoedas = PMMB.getNumPMMB();
double saldos = PMMB.getSaldoTotal();

PMMB.incNumPMMB();       //
PMMB.actSaldoTotal(valor);
```

- para que se consiga perceber o código é necessário que se siga a convenção que diz que as classes começam por letra maiúscula

Estruturas de Dados

- Como já vimos atrás, podemos criar conceitos mais complexos através do mecanismo de composição/agregação de objectos de classes mais simples:
  - a Turma a partir de Aluno
  - o Stand a partir de Veículo
  - etc.
- Mas para isso precisamos de ter colecções de objectos...

- Até ao momento apenas temos disponível a utilização de arrays

```
Aluno[] alunos = new Aluno[30];
Veiculo[] carros = new Veiculo[10];

for (int i=0; i<alunos.length && !encontrado; i++)
  if (alunos[i].getNota() == 20)
    encontrado = true;


for(Veiculo v: carros)
  System.out.println(v.toString());
```

- Esta é uma solução simples e testada, com a inconveniente de o tamanho da estrutura de dados ser estaticamente definido.

- Será que conseguimos ter uma estrutura de dados, baseada em arrays, que pudesse crescer de forma transparente para o utilizador?

  - (primeira solução) Sim, bastando para tal criarmos uma classe com esse comportamento

- Seja essa classe chamada de GrowingArray e, por comodidade, vamos utilizar instâncias de Circulo

- Que operações necessitamos:

  - adicionar um circulo (no fim e numa posição)
  - remover um círculo
  - ver se um circulo existe
  - dar a posição de um circulo na estrutura
  - dar número de elementos existentes

- Documentação com os métodos necessários:

Constructor Summary

`GrowingArray()`
`GrowingArray(int capacidade)`

Method Summary

void	`add(Circulo c)` Adiciona o elemento passado como parâmetro ao fim do array
void	`add(int indice, Circulo c)` Método que adiciona um elemento numa determinada posição, forçando a que os elementos à direita no array façam shift.
boolean	`contains(Circulo c)` Método que determina se um elemento está no array.
Circulo	`get(int indice)` Devolve o elemento que está na posição indicada.
int	`indexOf(Circulo c)` Método que determina o índice do array onde está localizada a primeira ocorrência de um objecto.
boolean	`isEmpty()` Método que determina se o array contém elementos, ou se está vazio.
boolean	`remove(Circulo c)` Remove a primeira ocorrência do elemento que é passado como parâmetro.
Circulo	`remove(int indice)` Remove do array o elemento que está na posição indicada no parâmetro.
void	`set(int indice, Circulo c)` Método que actualiza o valor de uma determinada posição do array.
int	`size()` Método que determina o tamanho do array de elementos.

- Declarações iniciais:

```
public class GrowingArray {

  private Circulo[] elementos;
  private int size;

  /**
   * variável que determina o tamanho inicial do array,
   * se for utilizado o construtor vazio.
   */
  private static final int capacidade_inicial = 20;

  public GrowingArray(int capacidade) {
    this.elementos = new Circulo[capacidade];
    this.size = 0;
  }

  public GrowingArray() {
    this(capacidade_inicial);
  }
```

- get de um elemento da estrutura de dados

```
/**
 * Devolve o elemento que está na posição indicada.
 *
 * @param  indice posição do elemento a devolver
 * @return o objecto que está na posição indicada no parâmetro
 * (deveremos ter atenção às situações em que a posição não existe)
 */

public Circulo get(int indice) {
   if (indice <= this.size)
       return this.elementos[indice];
   else
      return null; // ATENÇÃO!

}
```

- set de uma posição da estrutura

```
/**
 * Método que actualiza o valor de uma determinada posição do array.
 *
 * @param indice a posição que se pretende actualizar
 * @param c o circulo que se pretende colocar na estrutura de dados
 *
 */
public void set(int indice, Circulo c) {
  if (indice <= this.size) //não se permitem "espaços vazios"
    this.elementos[indice] = c;
}
```

- adicionar um elemento à estrutura de dados

```
/**
 * Adiciona o elemento passado como parâmetro ao fim do array
 *
 * @param c circulo que é adicionado ao array
 *
 */

public void add(Circulo c) {
  aumentaCapacidade(this.size + 1);
  this.elementos[this.size++] = c;
}
```

- método auxiliar que aumenta espaço

```
/**
 * Método auxiliar que verifica se o array alocado tem capacidade
 * para guardar mais elementos.
 * Por cada nova inserção, verifica se estamos a mais de metade
 * do espaço
 * alocado e,caso se verifique, aloca mais 1.5 de capacidade.
 *
 */

private void aumentaCapacidade(int capacidade) {
   if (capacidade > 0.5 * this.elementos.length) {
      int nova_capacidade = (int)(this.elementos.length * 1.5);
      this.elementos = Arrays.copyOf(this.elementos,nova_capacidade);
   }
}
```

```java
/**
 * Método que adiciona um elemento numa determinada posição,
 * forçando a
 * que os elementos à direita no array façam shift.
 * Tal como no método de set não são permitidos espaços.
 *
 * @param indice indice onde se insere o elemento
 * @param c circulo que será inserido no array
 *
 */

public void add(int indice, Circulo c) {
  if (indice <= this.size) {
    aumentaCapacidade(this.size+1);
    System.arraycopy(this.elementos, indice, this.elementos,
                     indice + 1, this.size - indice);
    this.elementos[indice] = c;
    this.size++;
  }
}
```

```java
/**
 * Remove do array o elemento que está na posição indicada no parâmetro.
 * Todos os elementos à direita do índice sofrem um deslocamento
 * para a esquerda.
 * @param indice índice do elemento a ser removido
 * @return o elemento que é removido do array. No caso do índice não
 * existir devolver-se-á null.
 */
public Circulo remove(int indice) {
    if (indice <= this.size) {
      Circulo c = this.elementos[indice];

      int deslocamento = this.size - indice - 1;
      if (deslocamento > 0)
        System.arraycopy(this.elementos, indice+1, this.elementos,
                         indice, deslocamento);
      this.elementos[--this.size] = null;
      return c;
    }
    else
      return null;
}
```

```java
 * Remove a primeira ocorrência do elemento que é passado como parâmetro.
 * Devolve true caso o array contenha o elemento, falso caso contrário.
 *
 * @param c círculo a ser removido do array (caso exista)
 * @return true, caso o círculo exista no array
*/
public boolean remove(Circulo c) {
   if (c != null) {
      boolean encontrado = false;
      for (int indice = 0; indice < this.size && !encontrado; indice++)
              if (c.equals(this.elementos[indice])) {
                  encontrado = true;
                  int deslocamento = this.size - indice - 1;
                  if (deslocamento > 0)
                    System.arraycopy(this.elementos, indice+1,
                        this.elementos, indice, deslocamento);
                  this.elementos[--this.size] = null;
              }
      return encontrado;
   }
   else
     return false;
}
```

```java
/**
 * Método que determina o índice do array onde está localizada a
 * primeira ocorrência de um objecto.
 *
 * @param c círculo de que se pretende determinar a posição
 * @return a posição onde o círculo se encontra. -1 caso não esteja no
 * array ou o círculo passado como parâmetro seja null.
 */

public int indexOf(Circulo c) {
   int posicao = -1;
   if (c != null) {
          boolean encontrado = false;
          for (int i = 0; i < this.size && !encontrado; i++)
              if (c.equals(this.elementos[i])) {
                  encontrado = true;
                  posicao = i;
              }
   }
   return posicao;
}
```

```java
/**
 * Método que determina se um elemento está no array.
 *
 * @param c círculo a determinar se está no array
 * @return true se o objecto estiver inserido na estrutura de dados,
 * false caso contrário.
 */
public boolean contains(Circulo c) {
  return indexOf(c) >= 0;
}
```

```java
/**
 * Método que determina se o array contém elementos, ou se está vazio.
 *
 * @return true se o array estiver vazio, false caso contrário.
 */

public boolean isEmpty() {
  return this.size == 0;
}
```

```java
public class TesteGA {
  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);

    GrowingArray ga = new GrowingArray(10);
    ga.add(c1.clone());
    ga.add(c2.clone());
    ga.add(c3.clone());

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

  }
}
```

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;
  - ...

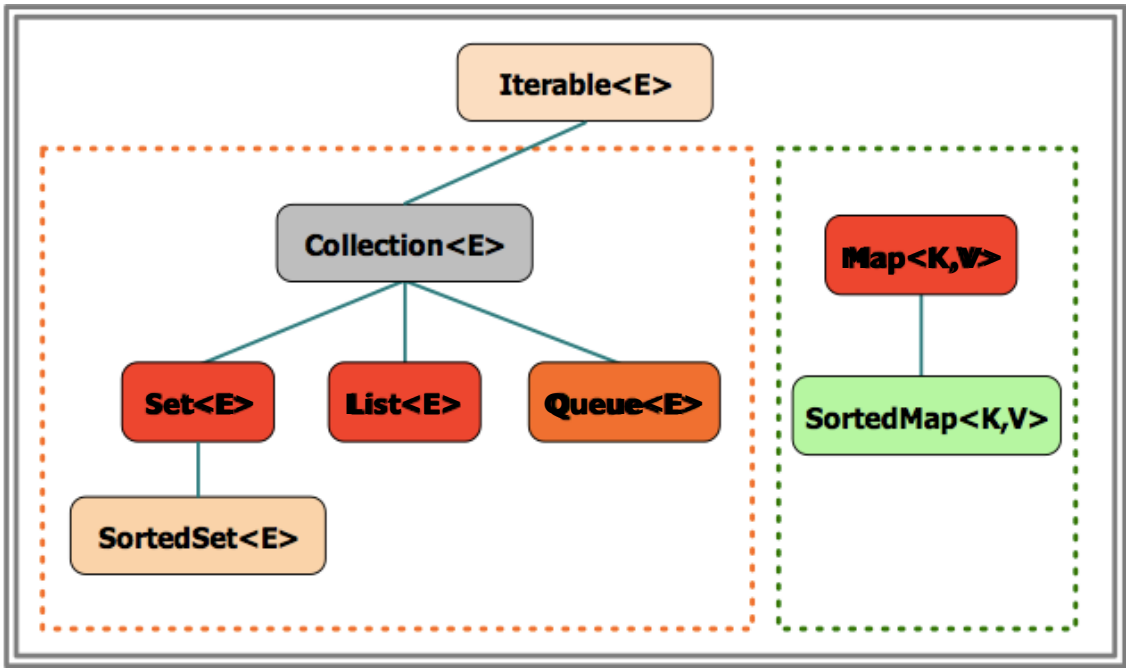
- Ao fazer-se ArrayList<Aluno> passa a ser o compilador a testar, e validar, que só são utilizados objectos do tipo Aluno no arraylist.
  - isto dá uma segurança adicional aos programas, pois em tempo de execução não teremos erros de compatibilidade de tipos
  - os tipos de dados são verificados em tempo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

JCF

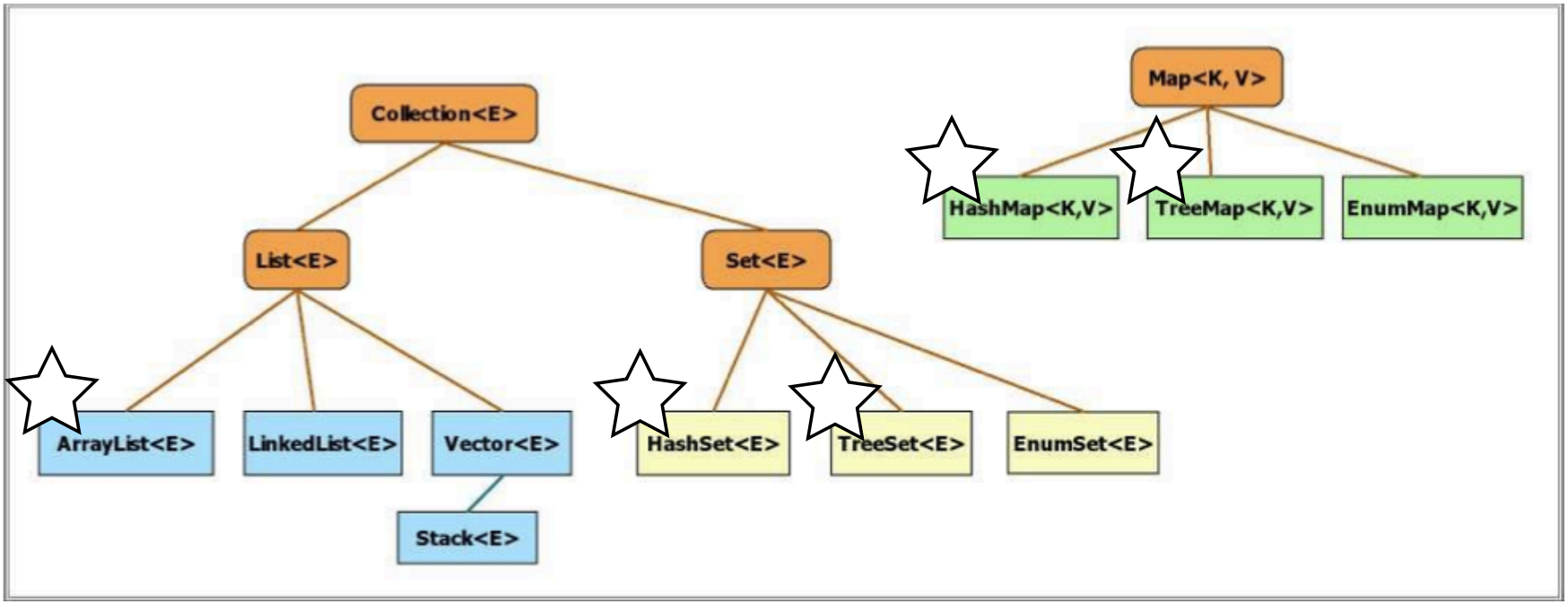
- O JCF (Java Collections Framework) agrupa as várias classes genéricas que correspondem às implementações de referência de:
  - Listas: API de **List<E>**
  - Conjuntos: API de **Set<E>**
  - Correspondências unívocas: API de **Map<K,V>**

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());

  }
}
```