

Programação Orientada por Objetos

Genéricos

Prof. José Cordeiro,

Prof. Cédric Grueau,

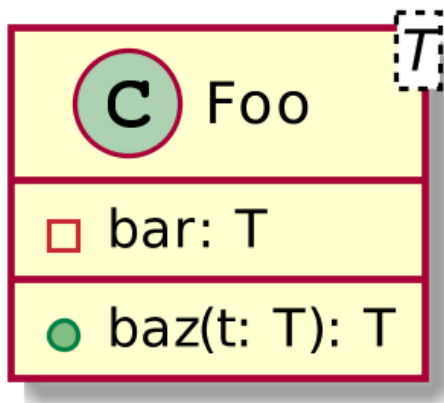
Prof. Laercio Júnior

Departamento de Sistemas e Informática

Escola Superior de Tecnologia de Setúbal – Instituto Politécnico de Setúbal

2019/2020

- ❑ Sessão 1: Exemplo de Listas
- ❑ Sessão 2: Exemplo com Listas de Objetos
- ❑ Sessão 3: Listas com Genéricos
- ❑ Sessão 4: Genéricos: Tópicos Avançados



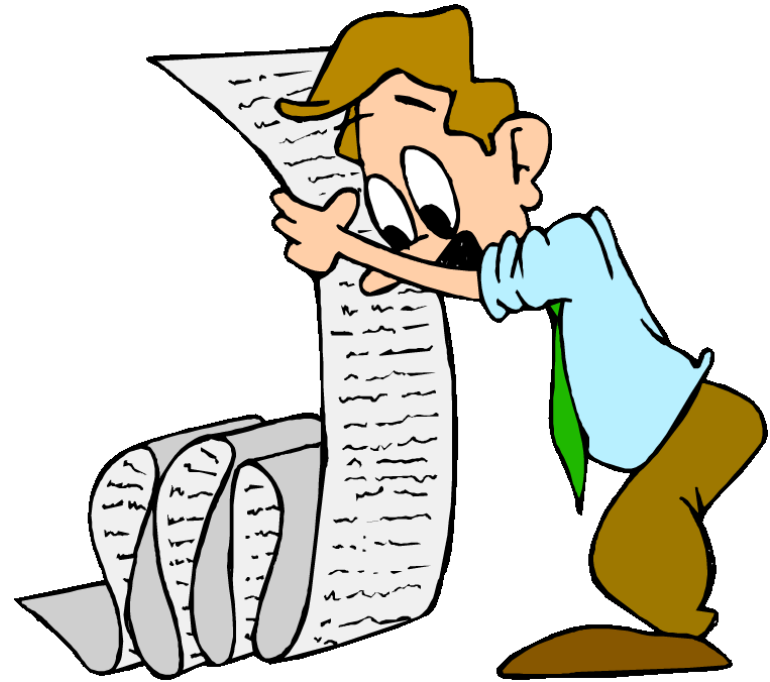
Módulo 5 – Genéricos

SESSÃO 1 – EXEMPLO DE LISTAS

Exemplo — Listas

□ Requisitos :

- Pretende-se criar um programa que irá guardar informação de nomes, de pessoas e de outro tipo de listas.
 - Deverá ser possível como é habitual em listas adicionar, alterar, remover e listar os elementos das listas.
 - Como restrição não será possível utilizar as classes de coleção do Java.



Exemplo — Listas

□ Classe **ListOfNames**

```
public class ListOfNames {
```

```
    private static final int DEFAULT_SIZE = 5;
```

```
    private String[] values;
```

```
    private int totalValues;
```

```
    public ListOfNames() {
```

```
        values = new String[DEFAULT_SIZE];
```

```
        totalValues = 0;
```

```
    }
```

```
// continua...
```

Tamanho do array

Nomes

Número de elementos no array

**Vamos guardar os
nomes num *array*.**

Exemplo — Listas

❑ Classe **ListOfNames** – métodos **add** e **remove**

```
public void add(String element) {  
    if (totalValues == values.length) {  
        String[] newValues = new String[values.length * 2];  
        for (int i = 0; i < values.length; i++) {  
            newValues[i] = values[i];  
        }  
        values = newValues;  
    }  
    values[totalValues++] = element;  
}  
  
public boolean remove(int position) {  
    if ((position >= 0) && (position < totalValues)) {  
        for (int i = position; i < totalValues - 1; i++) {  
            values[i] = values[i + 1];  
        }  
        totalValues--;  
        return true;  
    } else {  
        return false;  
    }  
}
```

O array cresce
automaticamente
quando atinge o limite

**Adicionar e
remover nomes
do array.**

Exemplo – Listas

❑ Classe **ListOfNames** – métodos **get**, **set** e **size**

```
public String get(int position) {  
    if ((position >= 0) && (position < totalValues)) {  
        return values[position];  
    } else {  
        return null;  
    }  
}
```

Alterar um nome

```
public boolean set(int position, String element) {  
    if ((position >= 0) && (position < totalValues)) {  
        values[position] = element;  
        return true;  
    } else {  
        return false;  
    }  
}
```

**Obter e alterar
nomes do array.**

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

Total de nomes no *array*

Exemplo — Listas

❑ Classe **ListOfNames** – métodos **capacity** e **toString**

```
public int capacity() {  
    return values.length;  
}
```

Capacidade do *array*

```
@Override  
public String toString() {  
    String result = "[";  
    boolean first = true;  
    for (int i = 0; i < totalValues; i++) {  
        if (first) {  
            first = false;  
        } else {  
            result += ", ";  
        }  
        result += values[i];  
    }  
    result += "];"  
    return result;  
}
```

**Listar os nomes
do array.**

Escreve os nomes
separados por vírgulas

Exemplo — Listas

❑ Classe ListOfNames – utilização

```
public static void testListOfNames() {  
    ListOfNames names = new ListOfNames();  
    System.out.println("No início: capacity=" + names.capacity());  
    names.add("Bruno");  
    names.add("Fausto");  
    names.add("José");  
    names.add("Rui");  
    names.add("Patricia");  
    names.add("Joaquim");  
    System.out.println("Depois de inseridos os elementos: capacity=" + names.capacity());  
    System.out.println("names=" + names);  
    System.out.println("size=" + names.size());  
    System.out.println("names[4]=" + names.get(4));  
    System.out.println("names[10]=" + names.get(10));  
    names.remove(4);  
    System.out.println("Depois de remove(4): names=" + names);  
    System.out.println("size=" + names.size());  
    names.set(4, "Silva");  
    System.out.println("Depois de set(4): names=" + names);  
}
```

Exemplo – Listas

❑ Classe ListOfNames – utilização

```
public static void testListOfNames() {  
    ListOfNames names = new ListOfNames();  
    System.out.println("No início: capacity=" + names.capacity());  
    names.add("Bruno");  
    names.add("Fausto");  
    names.add("José");  
    names.add("Rui");  
    names.add("Patricia");  
    names.add("Joaquim");  
    System.out.println("Depois de inseridos os elementos: capacity=" + names.capacity());  
    System.out.println("names=" + names);  
    System.out.println("size=" + names.size());  
    System.out.println("names[4]=" + names.get(4));  
    System.out.println("names[10]=" + names.get(10));  
    names.remove(4);  
    System.out.println("Depois de remove(4): names=" + names);  
    System.out.println("size=" + names.size());  
    names.set(4, "Silva");  
    System.out.println("Depois de set(4): names=" + names);  
}
```

NOMES:

No início: capacity=5

Depois de inseridos os elementos: capacity=10

nomes=[Bruno, Fausto, José, Rui, Patricia, Joaquim]

size=6

nomes[4]=Patricia

nomes[10]=null

Depois de remove(4): nomes=[Bruno, Fausto, José, Rui, Joaquim]

size=5

Depois de set(4): nomes=[Bruno, Fausto, José, Rui, Silva]

□ Classe **Person**

```
public class Person {  
    private int age;  
    private String name;  
  
    public Person(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    public int getAge() {  
        return age;  
    }  
  
    public void setAge(int age) {  
        this.age = age;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    @Override  
    public String toString() {  
        return name + " (" + age + " anos)";  
    }  
}
```

Exemplo — Listas

❑ Classe **ListOfPerson**

```
public class ListOfPerson {
```

```
    private static final int DEFAULT_SIZE = 5;
```

```
    private Person[] values;
```

```
    private int totalValues;
```

```
    public ListOfPerson() {
```

```
        values = new Person[DEFAULT_SIZE];
```

```
        totalValues = 0;
```

```
    }
```

```
// continua...
```

Tamanho do *array*

Pessoas

Número de elementos no *array*

**Vamos guardar as
pessoas num *array*
como foi feito com
os nomes**

Exemplo – Listas

❑ Classe **ListOfPerson** – métodos **add** e **remove**

```
public void add(Person element) {
    if (totalValues == values.length) {
        Person[] newValues = new Person[values.length * 2];
        for (int i = 0; i < values.length; i++) {
            newValues[i] = values[i];
        }
        values = newValues;
    }
    values[totalValues++] = element;
}

public boolean remove(int position) {
    if ((position >= 0) && (position < totalValues)) {
        for (int i = position; i < totalValues - 1; i++) {
            values[i] = values[i + 1];
        }
        totalValues--;
        return true;
    } else {
        return false;
    }
}
```

**Adicionar e remover
pessoas do array.**

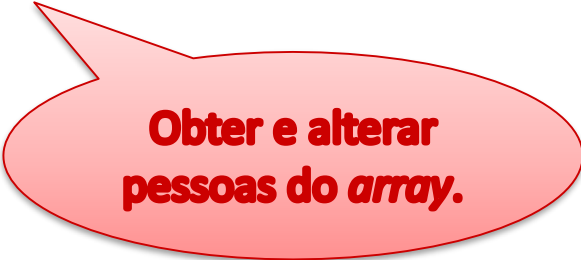
Exemplo – Listas

❑ Classe **ListOfPerson** – métodos **get**, **set** e **size**

```
public Person get(int position) {
    if ((position >= 0) && (position < totalValues)) {
        return values[position];
    } else {
        return null;
    }
}

public boolean set(int position, Person element) {
    if ((position >= 0) && (position < totalValues)) {
        values[position] = element;
        return true;
    } else {
        return false;
    }
}

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




**Obter e alterar
pessoas do *array*.**

Exemplo — Listas

❑ Classe **ListOfPerson** – métodos **capacity** e **toString**

```
public int capacity() {  
    return values.length;  
}  
  
@Override  
public String toString() {  
    String result = "[";  
    boolean first = true;  
    for (int i = 0; i < totalValues; i++) {  
        if (first) {  
            first = false;  
        } else {  
            result += ", ";  
        }  
        result += values[i];  
    }  
    result += "];"  
    return result;  
}
```



**Listar as pessoas
do array.**

Exemplo – Listas

❑ Classe **ListOfPerson** – utilização

```
public static void testListOfPerson() {
    ListOfPerson persons = new ListOfPerson();
    System.out.println("No início: capacity=" + persons.capacity());
    persons.add(new Person("Maria", 28));
    persons.add(new Person("Manuel", 34));
    persons.add(new Person("Marta", 45));
    persons.add(new Person("Mauro", 53));
    persons.add(new Person("Miguel", 19));
    persons.add(new Person("Margarida", 26));
    System.out.println("Depois de inseridos os elementos: capacity=" + persons.capacity());
    System.out.println("persons=" + persons);
    System.out.println("size=" + persons.size());
    System.out.println("persons[4]=" + persons.get(4));
    System.out.println("persons[10]=" + persons.get(10));
    persons.remove(4);
    System.out.println("Depois de remove(4): persons=" + persons);
    System.out.println("size=" + persons.size());
    persons.set(4, new Person("Matos", 47));
    System.out.println("Depois de set(4): persons=" + persons);
}
```


Exemplo – Listas

❑ Classe **ListOfPerson** – utilização

```
public static void testListOfPerson() {
    ListOfPerson persons = new ListOfPerson();
    System.out.println("No início: capacity=" + persons.capacity());
    persons.add(new Person("Maria", 28));
    persons.add(new Person("Manuel", 34));
    persons.add(new Person("Marta", 45));
    persons.add(new Person("Mauro", 53));
    persons.add(new Person("Miguel", 19));
    persons.add(new Person("Margarida", 26));
    System.out.println("Depois de inseridos os elementos: capacity=" + persons.capacity());
    System.out.println("persons=" + persons);
    System.out.println("size=" + persons.size());
    System.out.println("persons[4]=" + persons.get(4));
    System.out.println("persons[10]=" + persons.get(10));
    persons.remove(4);
    System.out.println(
        "Depois de remove(4): persons=" + persons);
    System.out.println("size=" + persons.size());
    persons.set(4, new Person("Matos", 47));
    System.out.println(
        "Depois de set(4): persons=" + persons);
}
```

PESSOAS:

No início: capacity=5

Depois de inseridos os elementos: capacity=10

peessoas=[Maria (28 anos), Manuel (34 anos), Marta (45 anos), Mauro (53 anos), Miguel (19 anos), Margarida (26 anos)]

size=6

peessoas[4]=Miguel (19 anos)

peessoas[10]=null

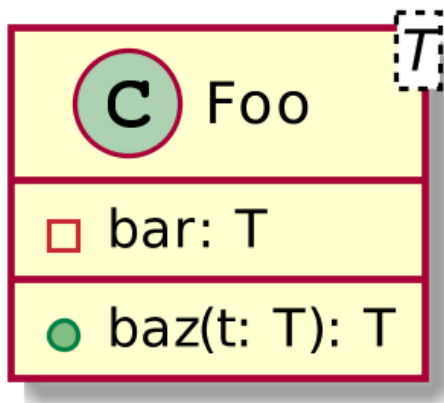
Depois de remove(4): peessoas=[Maria (28 anos), Manuel (34 anos), Marta (45 anos), Mauro (53 anos), Margarida (26 anos)]

size=5

Depois de set(4): peessoas=[Maria (28 anos), Manuel (34 anos), Marta (45 anos), Mauro (53 anos), Matos (47 anos)]

□ Análise das soluções

- Quando se consegue saber *à priori* o número de elementos que se pretende guardar, a utilização de um **array** é uma solução eficiente
- Quando não se sabe o número de elementos é preferível utilizar uma das **classes de coleção do Java**.
- Mas as duas soluções são muito parecidas...
 - Temos muita **duplicação de código!**
- Uma solução para o problema da duplicação de código pode ser criar uma única lista de **Object**

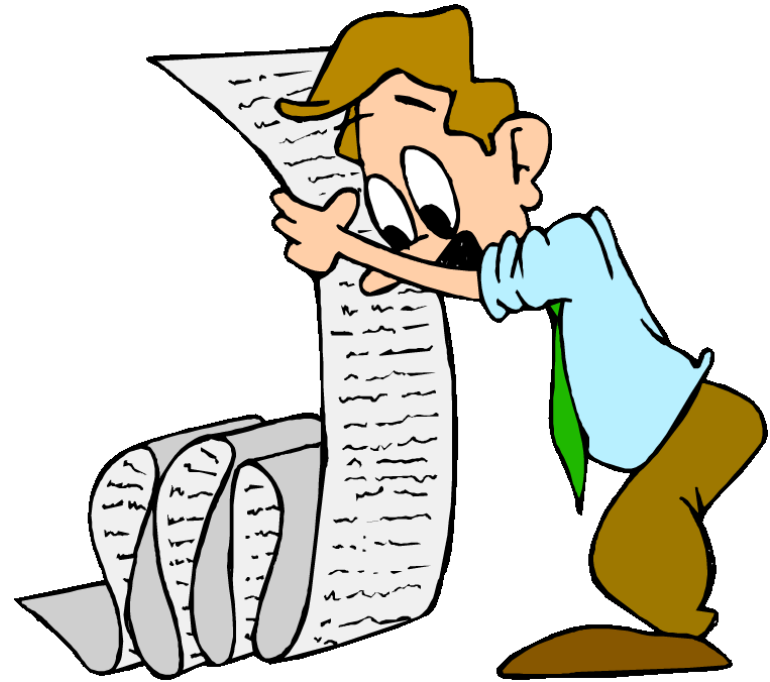


Módulo 5 – Genéricos

SESSÃO 2 – EXEMPLO COM LISTAS DE OBJETOS

□ Requisitos :

- Pretende-se criar um programa que irá guardar informação de nomes, de pessoas e de outro tipo de listas.
 - Deverá ser possível como é habitual em listas adicionar, alterar, remover e listar os elementos das listas.
 - Como restrição não será possível utilizar as classes de coleção do Java.



Exemplo — Listas

❑ Classe **ListOfObject**

```
public class ListOfObject {
```

```
    private static final int DEFAULT_SIZE = 5;  
    private Object[] values;  
    private int totalValues;
```

Qualquer tipo de objeto

```
    public ListOfObject() {  
        values = new Object[DEFAULT_SIZE];  
        totalValues = 0;  
    }
```

```
// continua...
```

**Vamos guardar os
objetos num *array***

Exemplo — Listas

❑ Classe **ListaOfObject** – métodos **add** e **remove**

```
public void add(Object element) {
    if (totalValues == values.length) {
        Object[] newValues = new Object[values.length * 2];
        for (int i = 0; i < values.length; i++) {
            newValues[i] = values[i];
        }
        values = newValues;
    }
    values[totalValues++] = element;
}

public boolean remove(int position) {
    if ((position >= 0) && (position < totalValues)) {
        for (int i = position; i < totalValues - 1; i++) {
            values[i] = values[i + 1];
        }
        totalValues--;
        return true;
    } else {
        return false;
    }
}
```

Exemplo — Listas

❑ Classe **ListaOfObject** – métodos **get**, **set** e **size**

```
public Object get(int position) {
    if ((position >= 0) && (position < totalValues)) {
        return values[position];
    } else {
        return null;
    }
}

public boolean set(int position, Object element) {
    if ((position >= 0) && (position < totalValues)) {
        values[position] = element;
        return true;
    } else {
        return false;
    }
}

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

❑ Classe **ListaOfObject** – métodos **capacity** e **toString**

```
public int capacity() {  
    return values.length;  
}
```

@Override

```
public String toString() {  
    String result = "[";  
    boolean first = true;  
    for (int i = 0; i < totalValues; i++) {  
        if (first) {  
            first = false;  
        } else {  
            result += ", ";  
        }  
        result += values[i];  
    }  
    result += "];"  
    return result;  
}
```


Exemplo – Listas

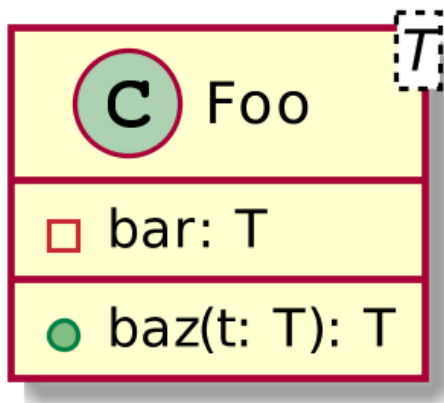
- ❑ Classe `ListaOfObject` – **utilização** com uma lista de Pessoas

```
public static void testListOfObject() {  
    ListOfObject objects = new ListOfObject();  
    System.out.println("No início: capacity=" + objects.capacity());  
    objects.add("Bruno");  
    objects.add("Fausto");  
    objects.add("José");  
    objects.add(new Person("Mauro", 53));  
    objects.add(new Person("Miguel", 19));  
    objects.add(new Person("Margarida", 26));  
    System.out.println("Depois de inseridos os elementos: capacity=" + objects.capacity());  
    System.out.println("objects=" + objects);  
    System.out.println("size=" + objects.size());  
    System.out.println("objects[4]=" + objects.get(4));  
    System.out.println("objects[10]=" + objects.get(10));  
    objects.remove(4);  
    System.out.println("Depois de remove(4): objects=" + objects);  
    System.out.println("size=" + objects.size());  
    objects.set(4, new Person("Matos", 47));  
    System.out.println("Depois de set(4): objects=" + objects);  
}
```

O código neste caso é semelhante ao anterior para a lista de pessoas. Mas...

Exemplo Listas

- Análise da solução **ListaOfObject**
 - Embora a solução com a classe **ListOfObject** seja semelhante existem alguns problemas.
 - A colocação de elementos na lista faz-se como anteriormente tirando partido do princípio da substituição
 - `objets.add(new Person("Margarida", 26));`
 - O método recebe **Object** como argumento e estamos a passar um objeto da classe **Person**
 - Quando se obtêm elementos da lista a situação é diferente
 - `Person person = (Person)objets.get(4);`
 - É necessário fazer um `cast` porque o método retorna **Object**
 - Outro problema é que o tipo de elementos guardado na lista não é verificado e podemos misturar objetos de diferentes classes
 - `objets.add("Margarida");`
 - Neste caso adicionámos uma **String**

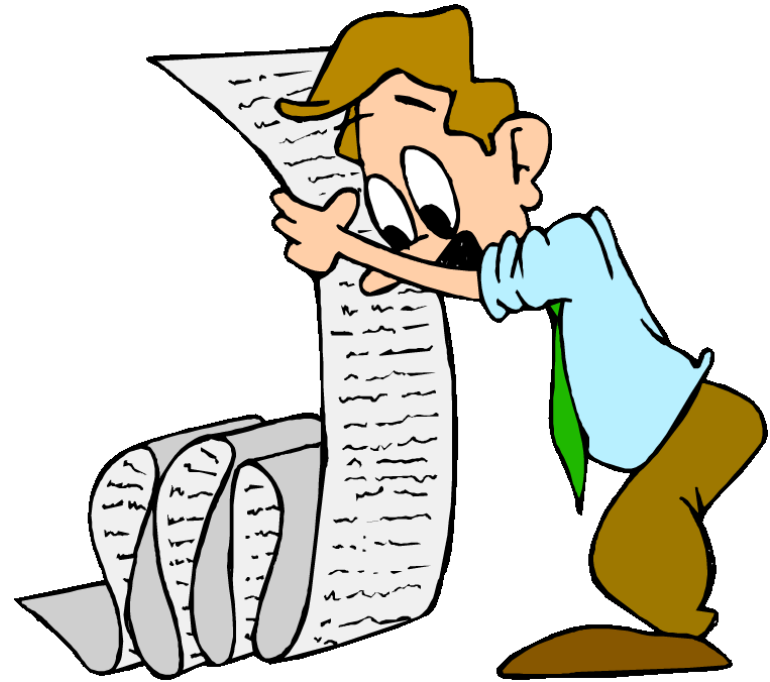


Módulo 5 – Genéricos

SESSÃO 3 – EXEMPLO LISTAS COM GENÉRICOS

□ Requisitos :

- Pretende-se criar um programa que irá guardar informação de nomes, de pessoas e de outro tipo de listas.
 - Deverá ser possível como é habitual em listas adicionar, alterar, remover e listar os elementos das listas.
 - Como restrição não será possível utilizar as classes de coleção do Java.



Exemplo – Listas

□ Classe `ListaOfNames` versus `ListOfPerson`

```
public class ListOfName {
```

```
    private static final int DEFAULT_SIZE = 5;
    private String[] values;
    private int totalValues;
```

```
    public ListOfName() {
        values = new String[DEFAULT_SIZE];
        totalValues = 0;
    }
```

```
    public void add(String element) {
        if (totalValues == values.length) {
            String[] newValues =
                new String[values.length * 2];
            for (int i = 0; i < values.length; i++) {
                newValues[i] = values[i];
            }
            values = newValues;
        }
        values[totalValues++] = element;
    }
```

```
// continua...
```


```
public class ListOfPerson {
```

```
    private static final int DEFAULT_SIZE = 5;
    private Person[] values;
    private int totalValues;
```

```
    public ListOfPerson() {
        values = new Person[DEFAULT_SIZE];
        totalValues = 0;
    }
```

```
    public void add(Person element) {
        if (totalValues == values.length) {
            Person[] newValues =
                new Person[values.length * 2];
            for (int i = 0; i < values.length; i++) {
                newValues[i] = values[i];
            }
            values = newValues;
        }
        values[totalValues++] = element;
    }
```

```
// continua...
```



Mudam apenas
os tipos de dados

Exemplo – Listas

❑ Classe **ListOfNames** versus **ListOfPerson**

```
// continuação ListOfNames
public boolean remove(int position) {
    if ((position >= 0) && (position < totalValues)) {
        for (int i = position; i < totalValues - 1; i++) {
            values[i] = values[i + 1];
        }
        totalValues--;
        return true;
    } else {
        return false;
    }
}

public String get(int position) {
    if ((position >= 0) && (position < totalValues)) {
        return values[position];
    } else {
        return null;
    }
}

public boolean set(int position, String element) {
    if ((position >= 0) && (position < totalValues)) {
        values[position] = element;
        return true;
    } else {
        return false;
    }
}
// continua...
```



```
// continuação ListOfPerson
public boolean remove(int position) {
    if ((position >= 0) && (position < totalValues)) {
        for (int i=position; i<totalValues-1; i++) {
            values[i] = values[i + 1];
        }
        totalValues--;
        return true;
    } else {
        return false;
    }
}

public Person get(int position) {
    if ((position >= 0) && (position < totalValues)) {
        return values[position];
    } else {
        return null;
    }
}

public boolean set(int position, Person element) {
    if ((position >= 0) && (position < totalValues)) {
        values[position] = element;
        return true;
    } else {
        return false;
    }
}
// continua...
```

Exemplo — Listas

❑ Classe **ListOfNames** versus **ListOfPerson**

// continuação ListOfNames

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

public int capacity() {
    return values.length;
}

@Override
public String toString() {
    String result = "[";
    boolean first = true;
    for (int i = 0; i < totalValues; i++) {
        if (first) {
            first = false;
        } else {
            result += ", ";
        }
        result += values[i];
    }
    result += "]";
    return result;
}
```



// continuação ListOfPerson

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

public int capacity() {
    return values.length;
}

@Override
public String toString() {
    String result = "[";
    boolean first = true;
    for (int i = 0; i < totalValues; i++) {
        if (first) {
            first = false;
        } else {
            result += ", ";
        }
        result += values[i];
    }
    result += "]";
    return result;
}
```

E se pudéssemos
fornecer o tipo
de dados dentro
duma variável?

Tipos Genéricos

- Tipo Genérico (**Generic**) ou Tipo Parametrizado (**type parameters**).
 - Na **definição de uma classe, ou de um método**, é possível indicar (entre < >) um parâmetro que representa um tipo de dados.
- Este parâmetro será utilizado nos locais onde se colocaria o tipo de dados (indicação do tipo dos atributos, na lista de parâmetros dos métodos, na declaração de variáveis)

```
public class List<E> {  
    ...  
}
```

Parâmetro E

Parâmetro T

```
public static <T> void information(T t) {  
    System.out.println("T: " + t.getClass().getName());  
}
```

**Utilização do parâmetro T
como um tipo de dados**

Tipos Genéricos

- Tipo Genérico (**Generic**) convenção de nomes:
 - Por convenção, os **type parameter** são representados normalmente apenas por uma letra, que indica o que o tipo representa:
 - **E** - Element (utilizado regularmente nas coleções do Java)
 - **K** - Key
 - **N** - Number
 - **T** - Type
 - **V** - Value
 - **S, U, V** etc. – 2º, 3º, 4º tipos

Exemplo — Listas

□ Classe **List** Genérica

O tipo E é fornecido
quando se criam os objetos

```
public class List<E> {  
  
    private static final int DEFAULT_SIZE = 5;  
    private E[] values;  
    private int totalValues;  
  
    public List() {  
        values = (E[]) new Object[DEFAULT_SIZE];  
        totalValues = 0;  
    }  
    // continua...
```

Não é possível fazer
new E[...] em Java


- Em Java não é possível criar **arrays** de tipos genéricos. A solução é criar um **array de Object** e depois fazer o **cast** para o tipo genérico

Exemplo – Listas

□ Classe **List** Genérica – métodos **add** e **remove**

```
public void add(E element) {
    if (totalValues == values.length) {
        Object[] newValues = new Object[values.length * 2];
        for (int i = 0; i < values.length; i++) {
            newValues[i] = values[i];
        }
        values = (E[]) newValues;
    }
    values[totalValues++] = element;
}

public boolean remove(int position) {
    if ((position >= 0) && (position < totalValues)) {
        for (int i = position; i < totalValues - 1; i++) {
            values[i] = values[i + 1];
        }
        totalValues--;
        return true;
    } else {
        return false;
    }
}
```



**Adicionar e remover
elementos do tipo E
do array.**


Exemplo – Listas

❑ Classe **List** Genérica – métodos **get**, **set** e **size**

```
public E get(int position) {  
    if ((position >= 0) && (position < totalValues)) {  
        return values[position];  
    } else {  
        return null;  
    }  
}
```

```
public boolean set(int position, E element) {  
    if ((position >= 0) && (position < totalValues)) {  
        values[position] = element;  
        return true;  
    } else {  
        return false;  
    }  
}
```

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



**Obter e alterar
elementos do tipo E
do array**

❑ Classe **List** Genérica – métodos **capacity** e **toString**

```
public int capacity() {  
    return values.length;  
}  
  
@Override  
public String toString() {  
    String result = "[";  
    boolean first = true;  
    for (int i = 0; i < totalValues; i++) {  
        if (first) {  
            first = false;  
        } else {  
            result += ", ";  
        }  
        result += values[i];  
    }  
    result += "];"  
    return result;  
}
```

Exemplo — Listas

□ Classe **List** **<E>** – utilização

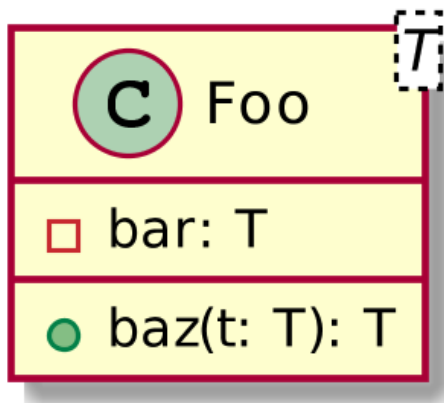
- No momento da utilização da classe indica-se o tipo pretendido:
 - `List<String> names = new List<String>();`
 - `List<Person> persons = new List<Person>();`
- A partir da Java SE 7 é possível omitir a indicação do tipo, sempre que o compilador consiga determiná-lo. Utilizando-se a chamada notação *diamante* **<>**:
 - `List<String> names = new List<>();`
 - `List<Person> persons = new List<>();`
- Na utilização dos métodos da classe não é necessário fazer qualquer modificação. Continua-se a poder utilizar elementos de classes derivadas:
 - Exemplo com uma classe **Worker** derivada de **Person**:
`persons.set(4, new Worker("Matos", 47));`

Métodos genéricos - Utilização

- A chamada a um **método genérico** deve indicar o tipo de dados a utilizar:

```
public class Generics {  
    ...  
    public static <T> void information(T t) {  
        System.out.println("T: " + t.getClass().getName());  
    }  
    ...  
}  
  
public static void main(String[] args) {  
    List<String> names = new List<>();  
    Generics.<List<String>>information(names);  
}
```

- Podemos omitir caso o compilador consiga determinar o tipo:
`Generics.information(names);`




Módulo 5 – Genéricos

SESSÃO 4 – GENÉRICOS: TÓPICOS AVANÇADOS

Múltiplos Tipos Parametrizados

- Podem ser indicados mais do que um **type parameter**:

```
public class Association<K, V> {  
    private K key;  
    private V value;  
  
    public Association(K key, V value) {  
        this.key = key;  
        this.value = value;  
    }  
  
    public K getKey() { return key; }  
    public void setKey(K key) { this.key = key; }  
    public V getValue() { return value; }  
    public void setValue(V value) { this.value = value; }  
}
```



**Representa uma
associação entre
elementos chave
(Key) e valor (Value)**

Limitar o Tipo Parametrizado

- É possível restringir o **type parameter** a um determinado tipo ou seus descendentes (através do uso de **extends**):

```
public class Association<K extends Number, V extends Person> {  
  
    private K key;  
    private V value;  
    ...  
}
```

- Desta forma podemos utilizar os métodos conhecidos do tipo:

```
@Override  
public String toString() {  
    return key + "-" + value.getName();  
}
```

Nota: **Number** tem como descendentes **AtomicInteger**, **AtomicLong**, **BigDecimal**, **BigInteger**, **Byte**, **Double**, **Float**, **Integer**, **Long**, **Short** ou outros que sejam definidos

Limitar o Tipo Parametrizado

- A **restrição** pode ser feita de **forma múltipla**:

```
public class A {  
    ...  
}  
public interface B {  
    ...  
}  
public interface C {  
    ...  
}  
  
public class D <T extends A & B & C> {  
    ...  
}
```

- A classe deverá ser indicada em primeiro lugar

Erro comum na percepção da Herança

- Apesar de se poder atribuir a uma lista de pessoas elementos que são de classes filhas (ex.: **Worker**). Não existe nenhuma relação entre **List<Person>** e **List<Worker>**, não sendo permitido a sua "mistura":

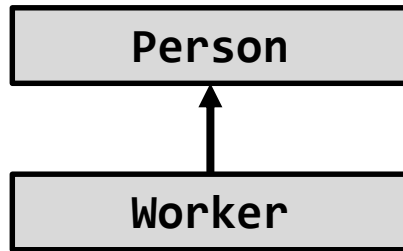
```
List<Person> persons = new List<>();
persons.add(new Person("Maria", 28));
persons.add(new Worker("Matos", 47));

List<Worker> workers = new List<>();
workers.add(new Worker("Mateus", 34));
workers.add(new Worker("Moureira", 53));
List<Person> error = workers;
```

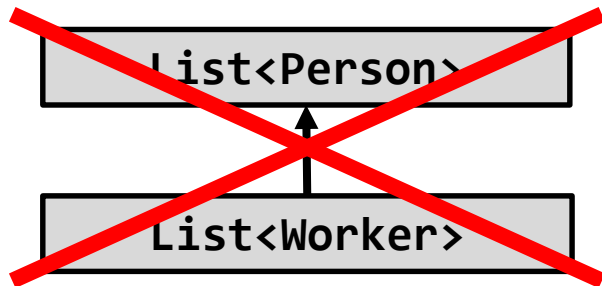
incompatible types: List<Worker> cannot be converted to List<Person>

Erro comum na percepção da Herança

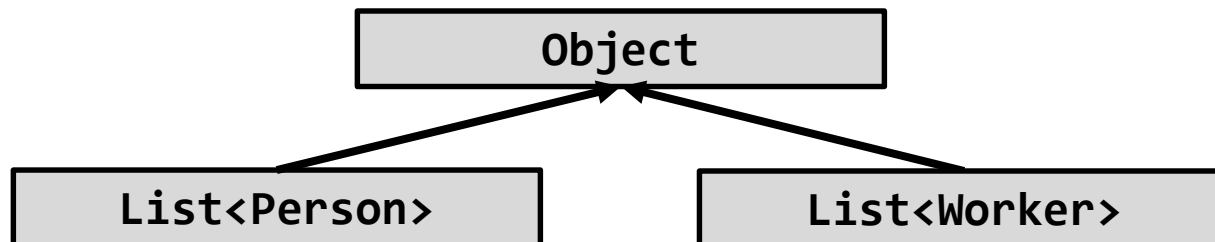
- **Worker** herda de **Person**:



- Mas **List<Worker>** não herda de **List<Person>**:



- Ambas **List<Worker>** e **List<Person>** herdam de **Object**:



Uso de ? (*wild-card*)

- ❑ O problema fica resolvido através da indicação de que o tipo de elementos da lista pode ser qualquer tipo (indicado através de ?) que herde de **Person**:

```
//List<Person> error = workers;
```

```
List<? extends Person> noError = workers;
```

- ❑ **? extends Person** indica qualquer tipo de herde de **Person** (inclusive). Desta forma indicamos não um tipo de lista mas sim uma "família de tipos de listas" que estão relacionados pela relação de herança dos seus elementos.
- ❑ Também é possível a notação **? super T**. Neste caso, seriam aceites elementos que fossem superclasses do tipo **T** (inclusive).

Limitações ao uso de Tipos Parametrizados

- ❑ Não é possível utilizar um tipo primitivo como substituição de um **type parameter**:

```
List<int> example = new List<>(); //ERRO!
```

- ❑ Devemos utilizar apenas tipos não primitivos:

```
List<Integer> example = new List<>(); //OK!
```

- ❑ Não é possível criar instâncias de um **type parameter**:

```
new E(); //new E[DEFAULT_SIZE];
```

- ❑ Não podem ser declarados atributos **static** cujo tipo sejam um **type parameter**:

```
private static E example;
```

Limitações ao uso de Tipos Parametrizados

- ❑ Não é possível fazer *cast* com um **type parameter**:
`example = (E)any;`
- ❑ Não é possível fazer **instanceof** com um **type parameter**:
`if (example instanceof E)`
- ❑ Não é possível criar **arrays** de Tipos Parametrizados:
`List<Integer>[] arrayLists = new List<Integer>[2];`
- ❑ Tipos Parametrizados não podem ser criados para fazer **throw** ou **catch**.
- ❑ Não pode ser feito polimorfismo de métodos que diferem apenas em Tipos Parametrizados:

```
public class Wrong {  
    public void print(List<String> listString) { }  
    public void print(List<Integer> listInteger) { }  
}
```


Resumindo

- ❑ O uso de **Tipos Parametrizados** permite definir classes e/ou métodos **genéricos** que envolvem tipos que apenas serão concretizados no momento da utilização
- ❑ Os Tipos parametrizados são, normalmente, **representados por uma letra** que indica o que o tipo representa.
- ❑ É possível **omitir o tipo envolvido** na utilização de métodos desde que o compilador o consiga determinar (poderá ser necessário usar a notação `<>`)
- ❑ Podem ser utilizados **múltiplos tipos parametrizados** e podemos limitar a gama de tipos a utilizar
- ❑ Pode ser necessário recorrer ao **uso de ?** para indicar relações entre tipos parametrizáveis.
- ❑ Existem algumas situações em que não é possível usar tipos parametrizáveis.