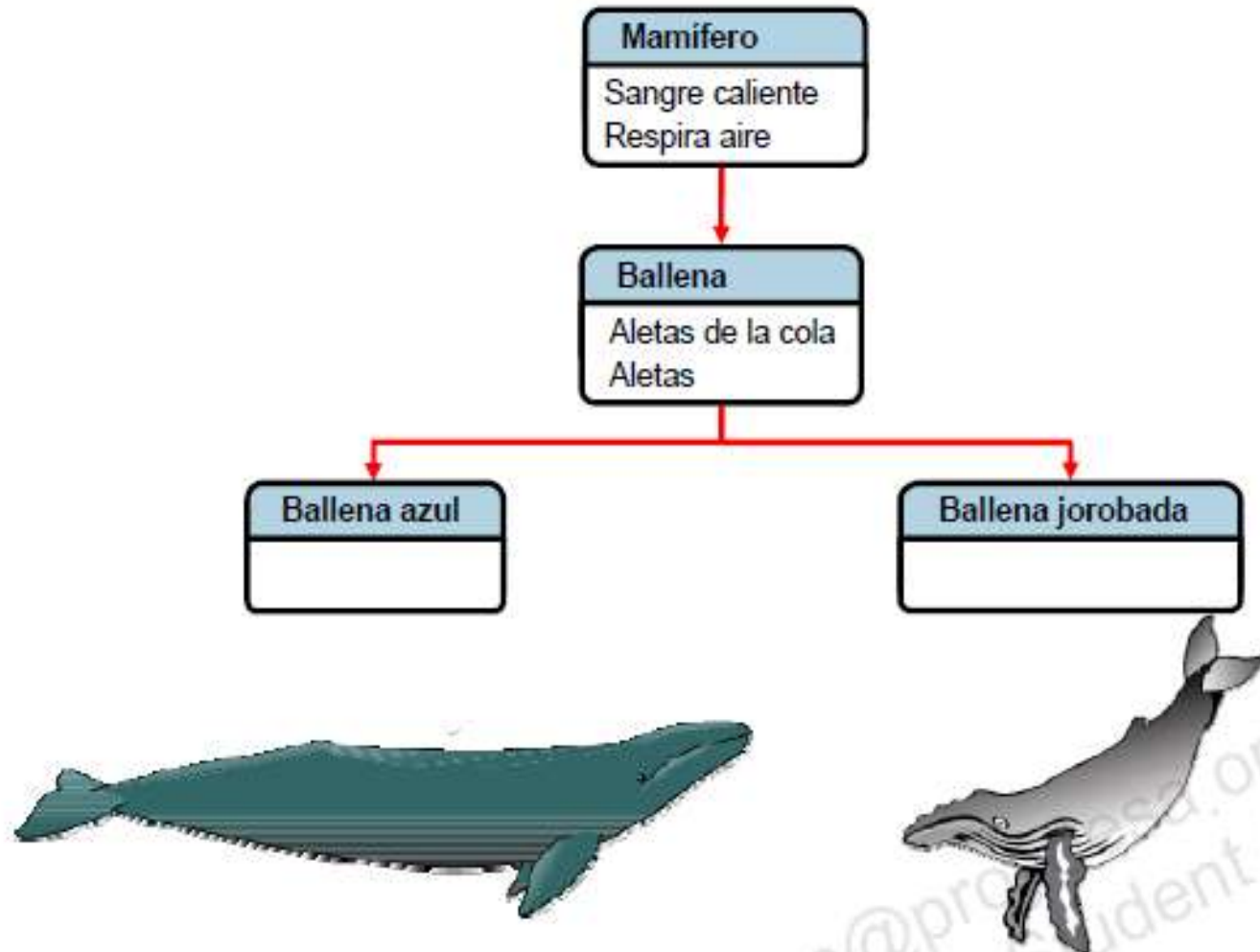


12

Uso de conceptos orientados a objetos avanzados

Jerarquías de clase



Comportamientos comunes

Shirt	Trousers
<code>getId()</code> <code>getPrice()</code> <code>getSize()</code> <code>getColor()</code> <code>getFit()</code>	<code>getId()</code> <code>getPrice()</code> <code>getSize()</code> <code>getColor()</code> <code>getFit()</code> <code>getGender()</code>
<code>setId()</code> <code>setPrice()</code> <code>setSize()</code> <code>setColor()</code> <code>setFit()</code>	<code>setId()</code> <code>setPrice()</code> <code>setSize()</code> <code>setColor()</code> <code>setFit()</code> <code>setGender()</code>
<code>display()</code>	<code>display()</code>

Duplicación de código

Shirt
<code>getId()</code>
<code>display()</code>
<code>getPrice()</code>
<code>getSize()</code>
<code>getColor()</code>
<code>getFit()</code>



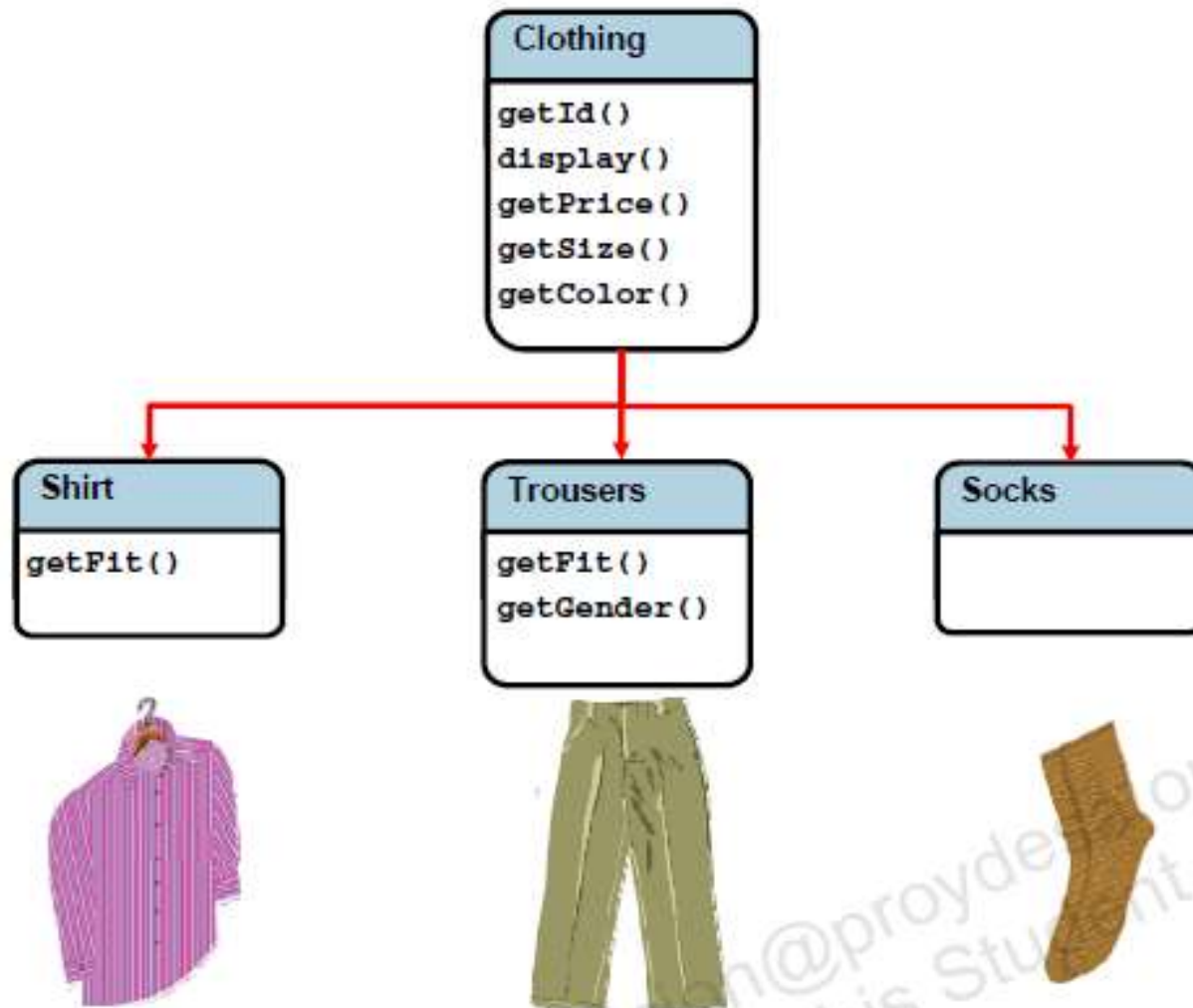
Trousers
<code>getId()</code>
<code>display()</code>
<code>getPrice()</code>
<code>getSize()</code>
<code>getColor()</code>
<code>getFit()</code>
<code>getGender()</code>



Socks
<code>getId()</code>
<code>display()</code>
<code>getPrice()</code>
<code>getSize()</code>
<code>getColor()</code>



Herencia



Sustitución de métodos de superclase

Puede que los métodos que existen en la superclase:

- No estén implantados en la subclase.
 - El método declarado en la superclase se utiliza en tiempo de ejecución.
- Estén implantados en la subclase.
 - El método declarado en la subclase se utiliza en tiempo de ejecución.

Superclase Clothing: 1

```
public class Clothing {  
    // Fields  
    private int itemID = 0; // Default ID for all clothing items  
    private String description = "-description required-"; // default  
    private char colorCode = 'U'; //'U' is Unset  
    private double price = 0.0; // Default price for all items  
  
    // Constructor  
    public Clothing(int itemID, String description, char colorCode,  
        double price) {  
        this.itemID = itemID;  
        this.description = description;  
        this.colorCode = colorCode;  
        this.price = price; }  
}
```


Superclase Clothing: 2

```
public void display() {  
    System.out.println("Item ID: " + getItemID());  
    System.out.println("Item description: " + description);  
    System.out.println("Item price: " + getPrice());  
    System.out.println("Color code: " + getColorCode());  
} // end of display method  
public String getDescription(){  
    return description;  
}  
public double getPrice() {  
    return price;  
}  
public int getItemID() {  
    return itemID;  
}
```


Superclase Clothing: 3

```
public char getColorCode() {  
    return colorCode;  
}  
public void setItemID(int itemID) {  
    this.itemID = itemID;  
}  
public void setDescription(String description) {  
    this.description = description;  
}  
public void setColorCode(char colorCode) {  
    this.colorCode = colorCode;  
}  
public void setPrice(double price) {  
    this.price = price;  
}
```

Declaración de una subclase

Sintaxis:

```
[class_modifier] class class_identifier extends superclass_identifier
```

Declaración de una subclase (palabras clave `extends`, `super` y `this`)

```
public class Shirt extends Clothing {
```

Se asegura de que Shirt hereda los miembros de Clothing.

```
    private char fit = 'U'; //'U' is Unset, other codes 'S', 'M', or 'L'
```

```
    public Shirt(int itemID, String description, char colorCode, double price, char fit) {
```

```
        super(itemID, description, colorCode, price);
```

`super` es una referencia a los métodos y atributos de la superclase.

```
        this.fit = fit;
```

`this` es una referencia a este objeto.

```
    }

    public char getFit() {
        return fit;
    }
```

```
    public void setFit(char fit) {
        this.fit = fit;
    }
}
```

Declaración de una subclase: 2

```
//This method overrides display in the Clothing superclass
public void display() {
    System.out.println("Shirt ID: " + getItemID());
    System.out.println("Shirt description: " + description);
    System.out.println("Shirt price: " + getPrice());
    System.out.println("Color code: " + getColorCode());
    System.out.println("Fit: " + getFit());
} // end of display method

// This method overrides the methods in the superclass
public void setColorCode(char colorCode) {

    ... include code here to check that correct codes used ...

    this.colorCode = colorCode;
}
}
```

Classes abstractas



=



=



=



=



Superclase abstracta Clothing: 1

```
public abstract class Clothing {  
    // Fields  
    private int itemID = 0; // default ID for all clothing items  
    private String description = "-description required-"; // default  
    private char colorCode = 'U'; // default color code for all items  
    private double price = 0.0; // default price for all items  
  
    // Constructor  
    public Clothing(int itemID, String description, char colorCode,  
        double price, int quantityInStock) {  
        this.itemID = itemID;  
        this.description = description;  
        this.colorCode = colorCode;  
        this.price = price;  
    }  
}
```

La palabra clave
abstract asegura
que la clase no se
puede instanciar.

Superclase abstracta Clothing: 2

```
public abstract char getColorCode() ;  
  
public abstract void setColorCode(char colorCode);  
  
... other methods not listed ...  
}
```

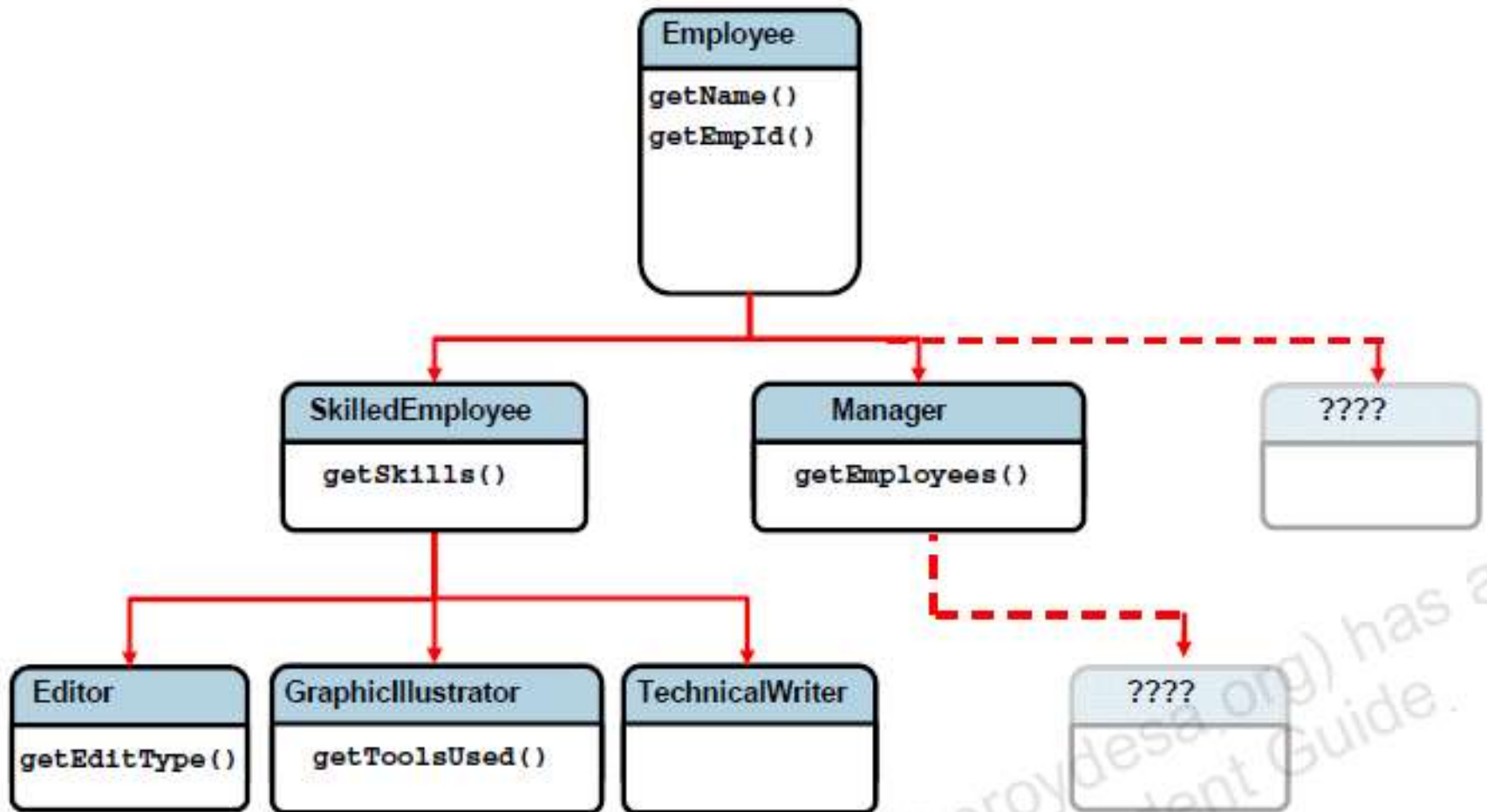
La palabra clave `abstract` asegura que estos valores se deben sustituir en la subclase.

Relaciones de superclases y subclases

Es muy importante tener en cuenta el uso más adecuado de la herencia:

- Utilice la herencia sólo cuando sea completamente válida o inevitable.
- Compruebe si es adecuada con la frase “es *un/una*”:
 - La frase “una camisa es un tipo de ropa” expresa un enlace de herencia válido.
 - La frase “un sombrero es un calcetín” expresa un enlace de herencia no válido.

Otro ejemplo de herencia



Tipos de referencia de superclase

Hasta ahora hemos analizado la clase utilizada como el tipo de referencia para el objeto creado:

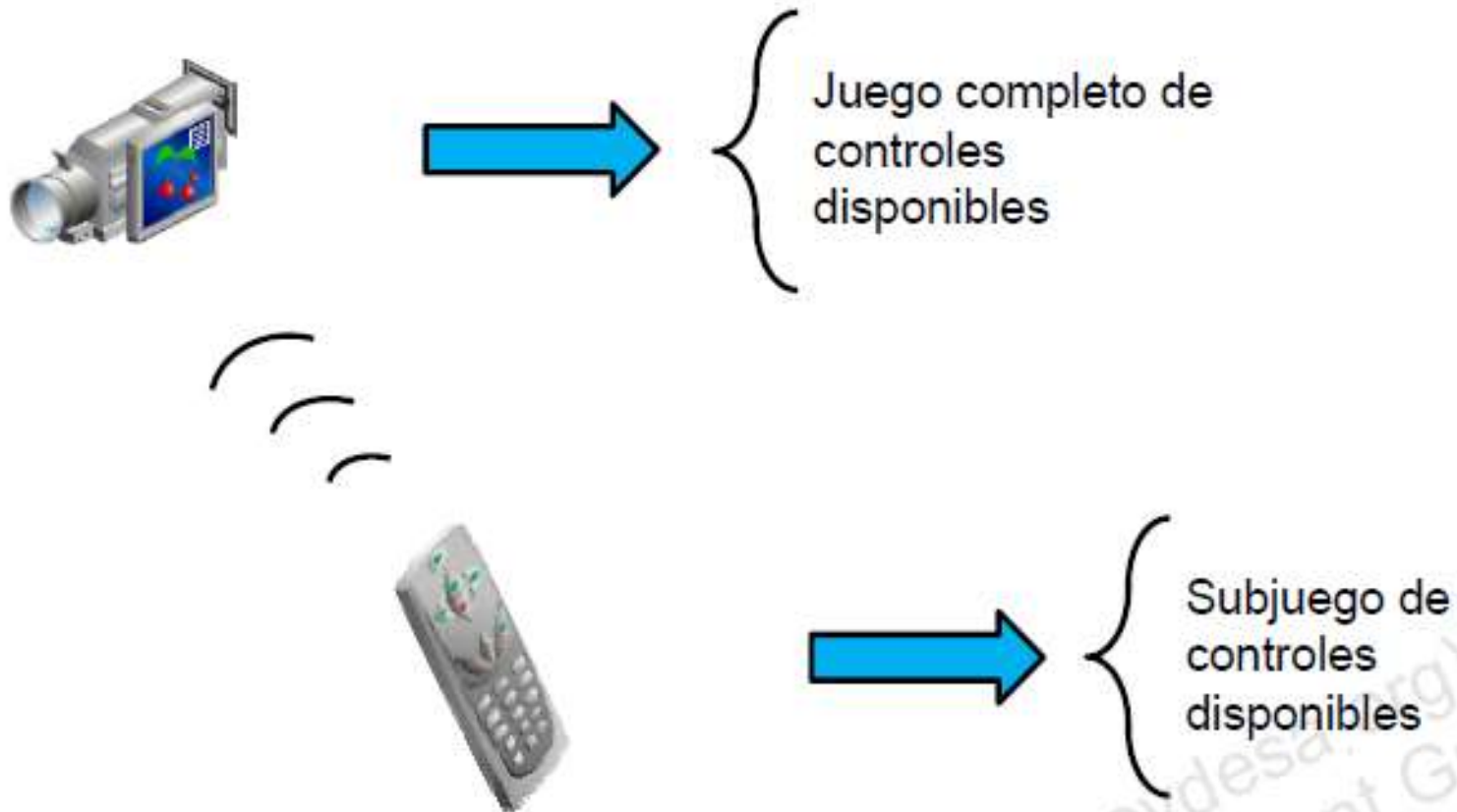
- Para utilizar la clase Shirt como el tipo de referencia para el objeto Shirt:
- Sin embargo, también puede utilizar la superclase como la referencia:

```
Shirt myShirt = new Shirt();
```

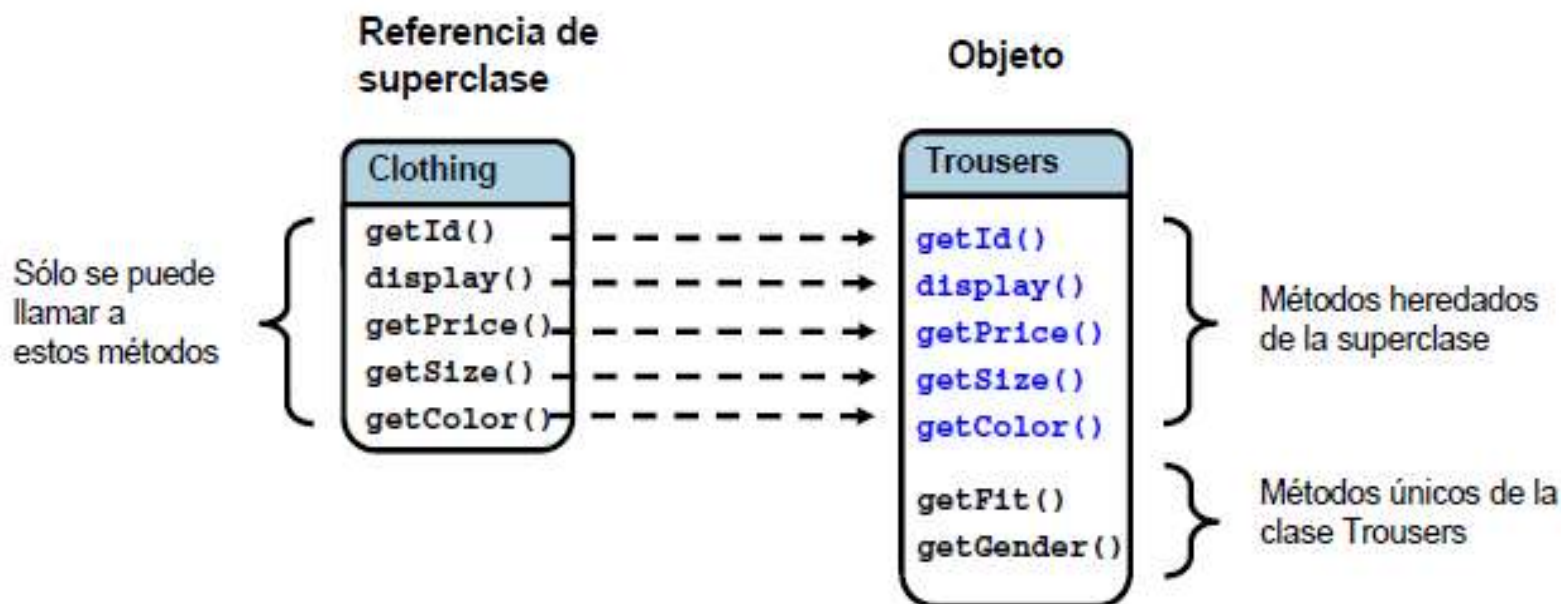
```
Clothing clothingItem1 = new Shirt();
```

```
Clothing clothingItem2 = new Trousers();
```

Acceso a funcionalidades de objeto



Acceso a métodos de clase desde la superclase



Conversión del tipo de referencia

Operación de conversión

La conversión cambia el tipo de referencia.

Referencia de superclase

Clothing

getId()
display()
getPrice()
getSize()
getColor()

Referencia de clase

Trousers

getId()
display()
getPrice()
getSize()
getColor()
getFit()
getGender()

Objeto

Trousers

getId()
display()
getPrice()
getSize()
getColor()
getFit()
getGender()

Métodos heredados de la superclase

Métodos únicos de la clase Trousers

Ahora se puede acceder a todos los métodos.

Conversión

```
Clothing cl = new Trousers(123, "Dress Trousers", 'B', 17.00, 4, 'S');  
cl.display();
```

```
//char fitCode = cl.getFit(); // This won't compile
```

```
char fitCode = ((Trousers)cl).getFit(); // This will compile
```

Los paréntesis de `cl` aseguran que la conversión se aplica a esta referencia.

Sintaxis de la conversión: el tipo que se va a convertir se coloca entre paréntesis, antes de la referencia que se va a convertir.

Operador instanceof

Posible error de conversión:

```
public static void displayDetails(Clothing cl) {  
  
    cl.display();  
    char fitCode = ((Trousers) cl).getFitCode();  
    System.out.println("Fit: " + fitCode);  
}
```

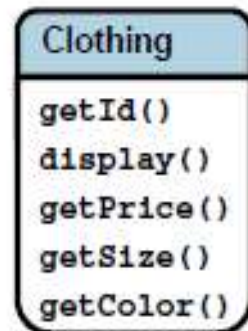
El operador instanceof se utiliza para asegurarse de que no existe ningún error de conversión:

```
public static void displayDetails(Clothing cl) {  
    cl.display();  
    if (cl instanceof Trousers) {  
        char fitCode = ((Trousers) cl).getFitCode();  
        System.out.println("Fit: " + fitCode);  
    }  
    else { // Take some other action }
```

El operador instanceof devuelve true si el objeto al que hace referencia cl es un objeto Trousers.

Llamadas a métodos polimórficos

Referencia de
superclase



Referencia de
clase



Independientemente del
tipo de referencia utilizado,
el método ejecutado está
en el objeto instanciado.

Objeto

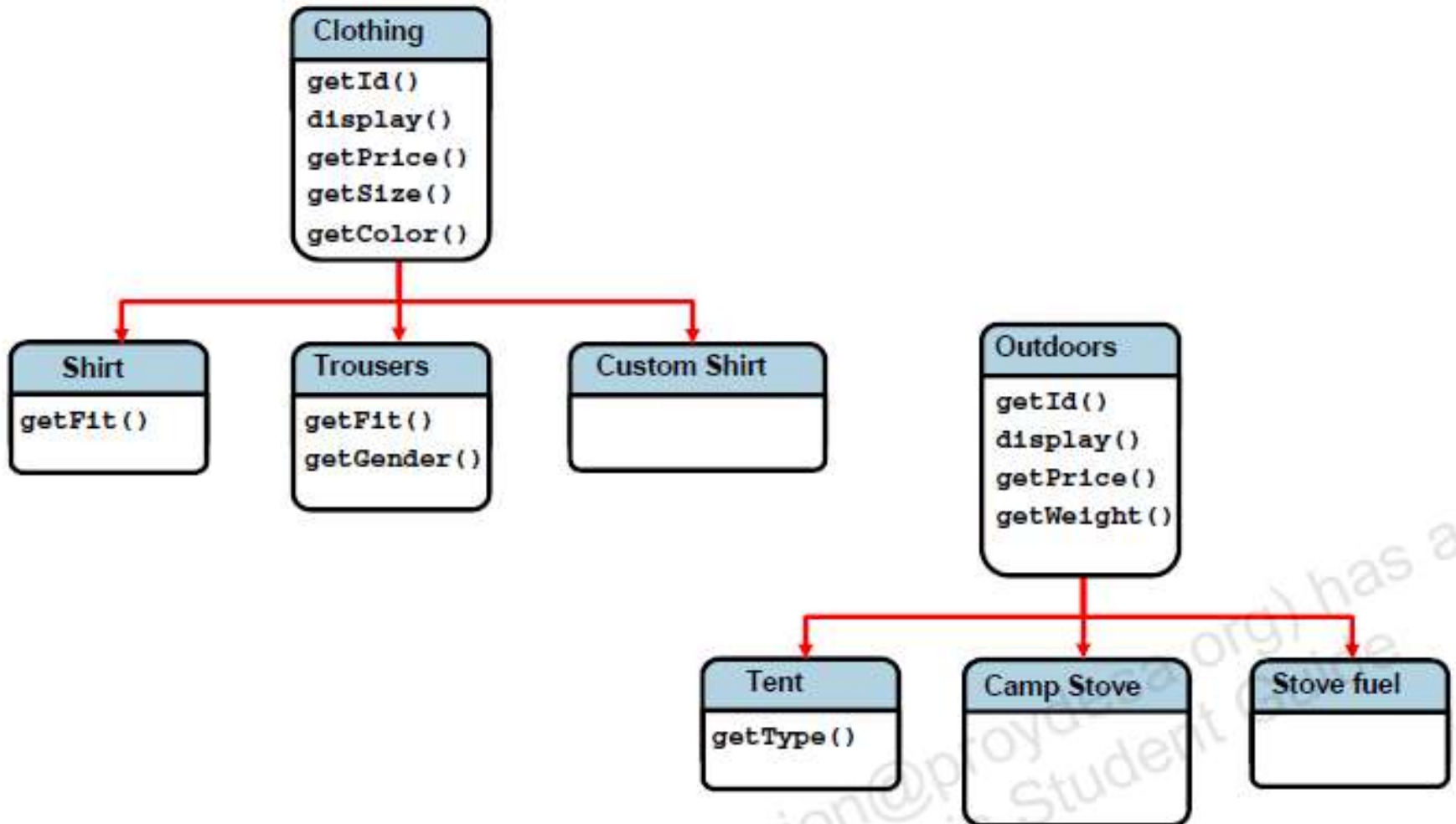


Métodos heredados
de la superclase

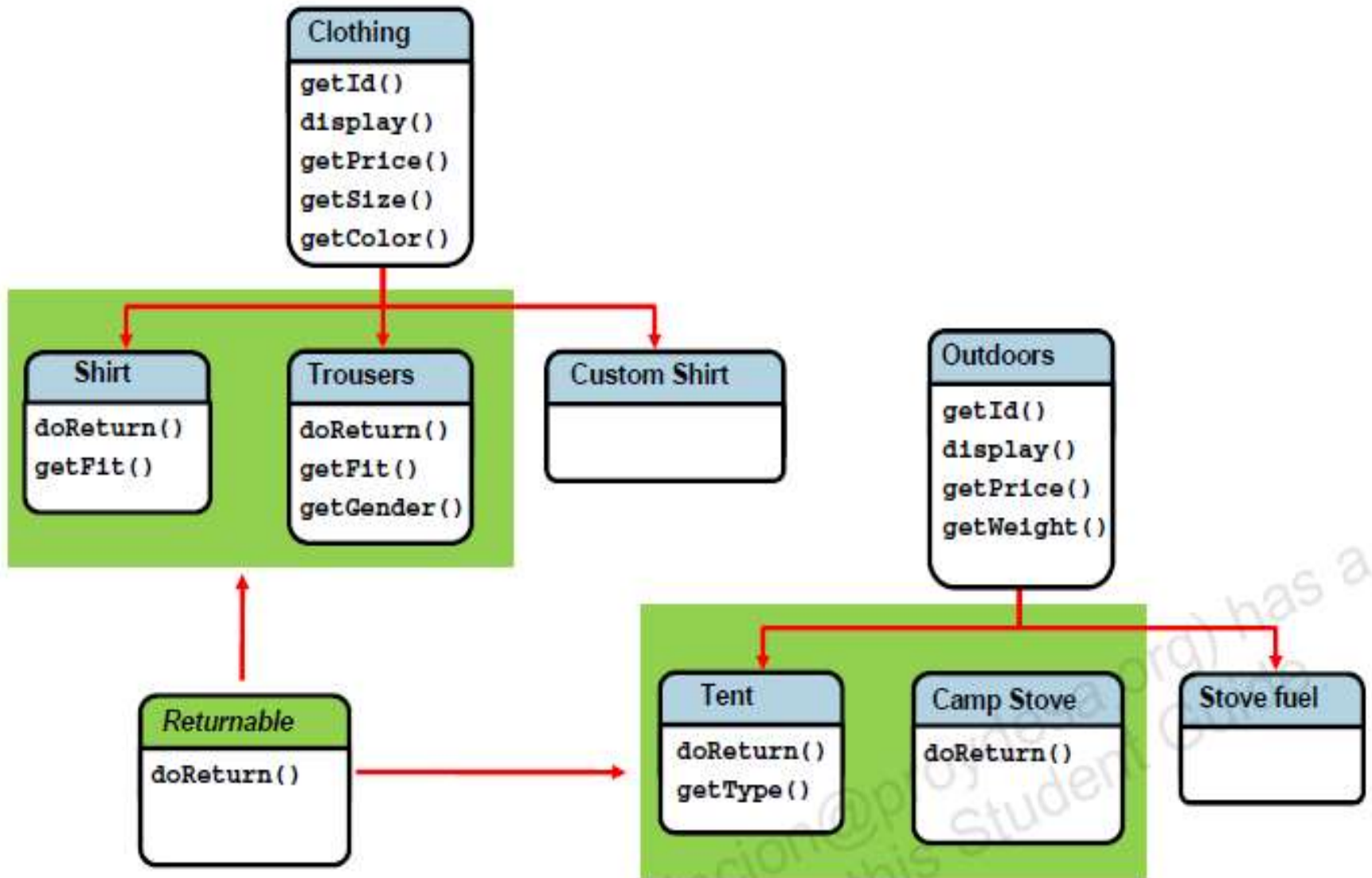
Métodos heredados
de la superclase
y sustituidos

Métodos únicos de la
clase Trousers

Varias jerarquías



Interfaces



Implantación de la interfaz Returnable

Interfaz Returnable

```
public interface Returnable {  
    public String doReturn();  
}
```

Al igual que un método abstracto, sólo tiene el stub de método.

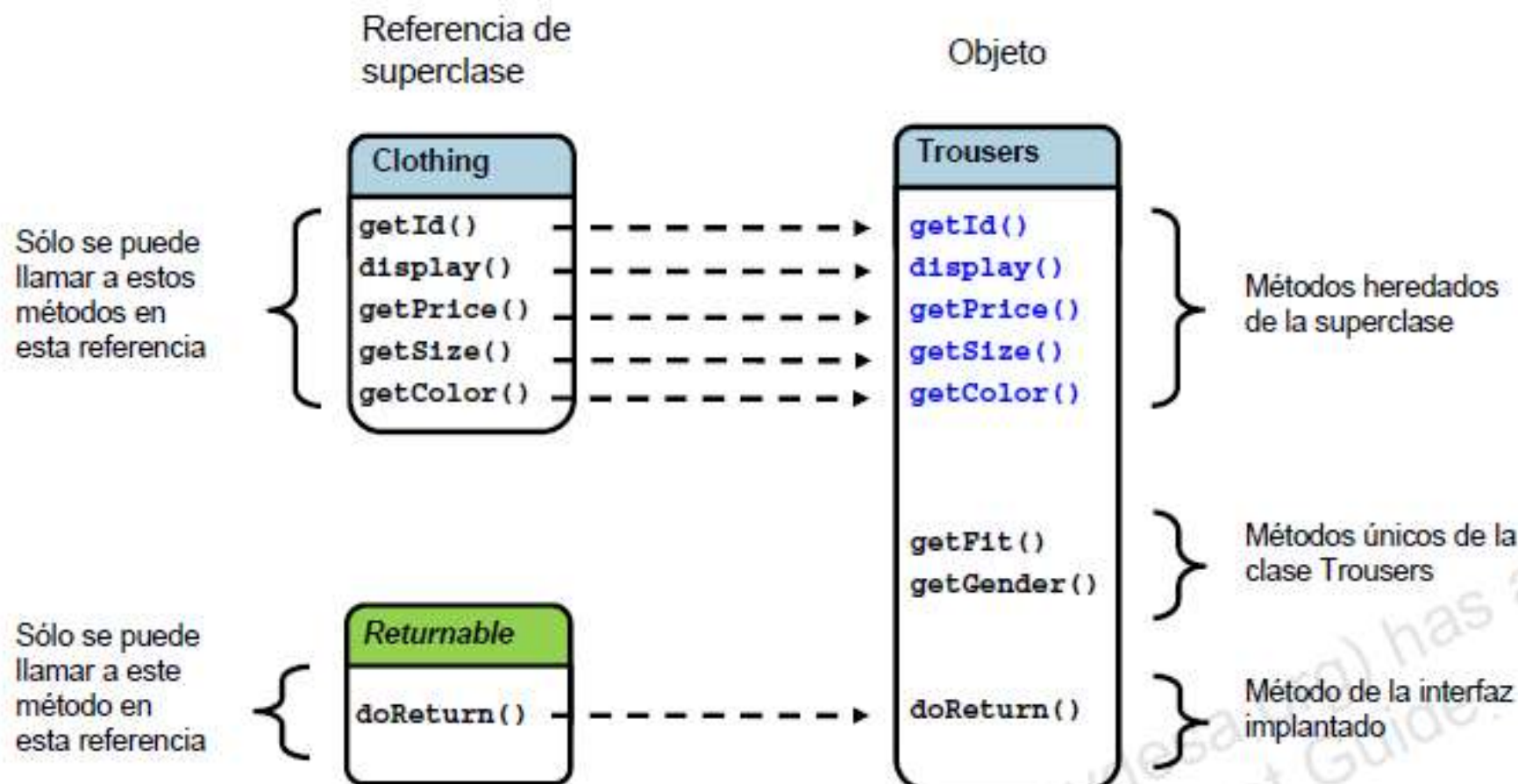
Asegura que Shirt debe implantar todos los métodos de Returnable.

Clase Shirt

```
public class Shirt extends Clothing implements Returnable {  
    public Shirt(int itemID, String description, char colorCode,  
        double price, char fit) {  
        super(itemID, description, colorCode, price);  
        this.fit = fit;  
    }  
    public String doReturn() {  
        // See notes below  
        return "Suit returns must be within 3 days";  
    }  
    ... < other methods not shown > ... } // end of class
```

Método declarado en la interfaz Returnable

Acceso a los métodos de objeto desde la interfaz



ArrayList

ArrayList se amplía desde AbstractList que, a su vez, se amplía desde AbstractCollection.

The screenshot shows the Java API documentation for the `ArrayList<E>` class. The page title is "Class ArrayList<E>". The inheritance hierarchy is shown as `java.lang.Object` → `java.util.AbstractCollection<E>` → `java.util.AbstractList<E>` → `java.util.ArrayList<E>`. The "All Implemented Interfaces" section lists `Serializable`, `Cloneable`, `Iterable<E>`, `Collection<E>`, `List<E>`, and `RandomAccess`. The "Direct Known Subclasses" section lists `AttributeList`, `RoleList`, and `RoleUnresolvedList`. The class declaration is `public class ArrayList<E> extends AbstractList<E> implements List<E>, RandomAccess, Cloneable, Serializable`. A red box highlights the description: "Resizable-array implementation of the List interface. Implements all optional list operations, and permits all elements, including null. In addition to implementing the List interface, this class provides methods to manipulate the size of the array that is used internally to store the list. (This class is roughly equivalent to Vector, except that it is unsynchronized)." Callouts from the surrounding text blocks point to these specific parts of the documentation.

Overview Package **Class** Use Tree Deprecated Index Help

Prev Class Next Class Frames No Frames

Summary: Nested | Field | Constructor | Method Detail: Field | Constructor | Method

java.util

Class ArrayList<E>

java.lang.Object

`java.util.AbstractCollection<E>`
`java.util.AbstractList<E>`
`java.util.ArrayList<E>`

All Implemented Interfaces:
`Serializable`, `Cloneable`, `Iterable<E>`, `Collection<E>`, `List<E>`, `RandomAccess`

Direct Known Subclasses:
`AttributeList`, `RoleList`, `RoleUnresolvedList`

`public class ArrayList<E>
extends AbstractList<E>
implements List<E>, RandomAccess, Cloneable, Serializable`

Resizable-array implementation of the List interface. Implements all optional list operations, and permits all elements, including null. In addition to implementing the List interface, this class provides methods to manipulate the size of the array that is used internally to store the list. (This class is roughly equivalent to Vector, except that it is unsynchronized)

ArrayList implanta una serie de interfaces.

La interfaz List es la que se utiliza principalmente al trabajar con ArrayList.

Interfaz List

Overview Package **Class** Use Tree Deprecated Index Help

Prev Class Next Class Frames No Frames

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.util

Interface List<E>

Type Parameters:

- E** - the type of elements in this list

All Superinterfaces:

- Collection<E>, Iterable<E>

All Known Implementing Classes:

AbstractList, AbstractSequentialList, ArrayList, AttributeList, CopyOnWriteArrayList, LinkedList, RoleList, RoleUnresolvedList, Stack, Vector

```
public interface List<E>
    extends Collection<E>
```

An ordered collection (also known as a sequence). The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list.

Existen varias clases que implantan la interfaz List

Class Object

Overview Package **Class** Use Tree Deprecated Index Help

Prev Class Next Class Frames No Frames

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.util

Class ArrayList<E>

java.lang.Object

- java.util.AbstractCollection<E>
- java.util.AbstractList<E>
- java.util.ArrayList<E>

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

Attributes:

public class extends implements Serializable

Resizable and mutable: all elements can be manipulated. Vector, etc.

java.lang

Class Object

java.lang.Object

```
public class Object
```

Class Object is the root of the class hierarchy. Every class has Object as a superclass. All objects, including arrays, implement the methods of this class.

La clase Object es la clase base.

Llamada al método toString()

Se utiliza toString() de Object.

StringBuilder sustituye toString() de Object.

First hereda toString() de Object.

Second sustituye toString() de Object.

```
1 public class Main {
2     public static void main(String[] args) {
3
4         // Output an Object to the console
5         System.out.println(new Object());
6
7         // Output this StringBuilder object to the console
8         System.out.println(new StringBuilder("Some text for StringBuilder"));
9
10        //Output a class that does not override the toString() method
11        System.out.println(new First());
12
13        //Output a class that *does* override the toString() method
14        System.out.println(new Second());
15    }
16 }
```

Output - TestCode (run)

```
java.lang.Object@3e25a5
Some text for StringBuilder
First@15821f
This class named Second has overridden the toString() method of Object
BUILD SUCCESSFUL (total time: 1 second)
```

Salida de las llamadas al método toString() de cada objeto.