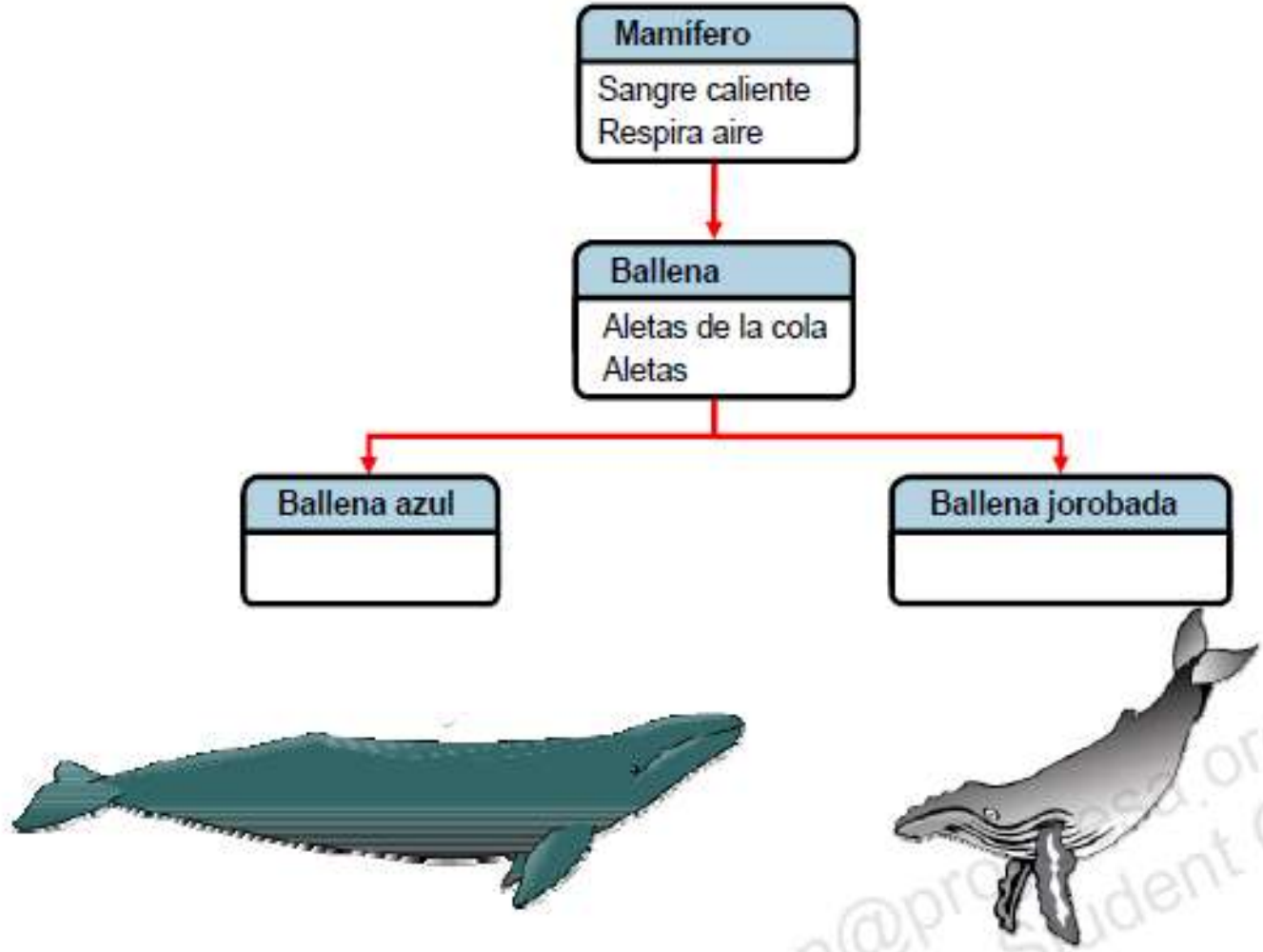
12

Uso de conceptos orientados a objetos avanzados

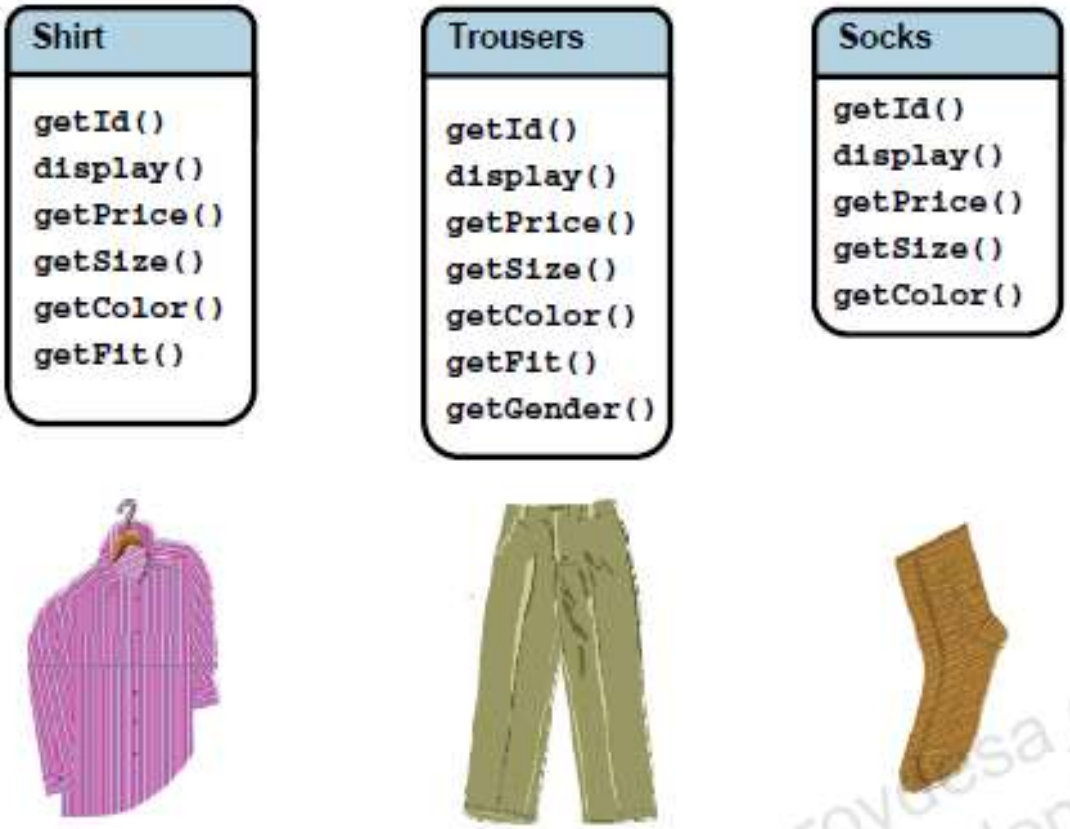
Jerarquías de clase



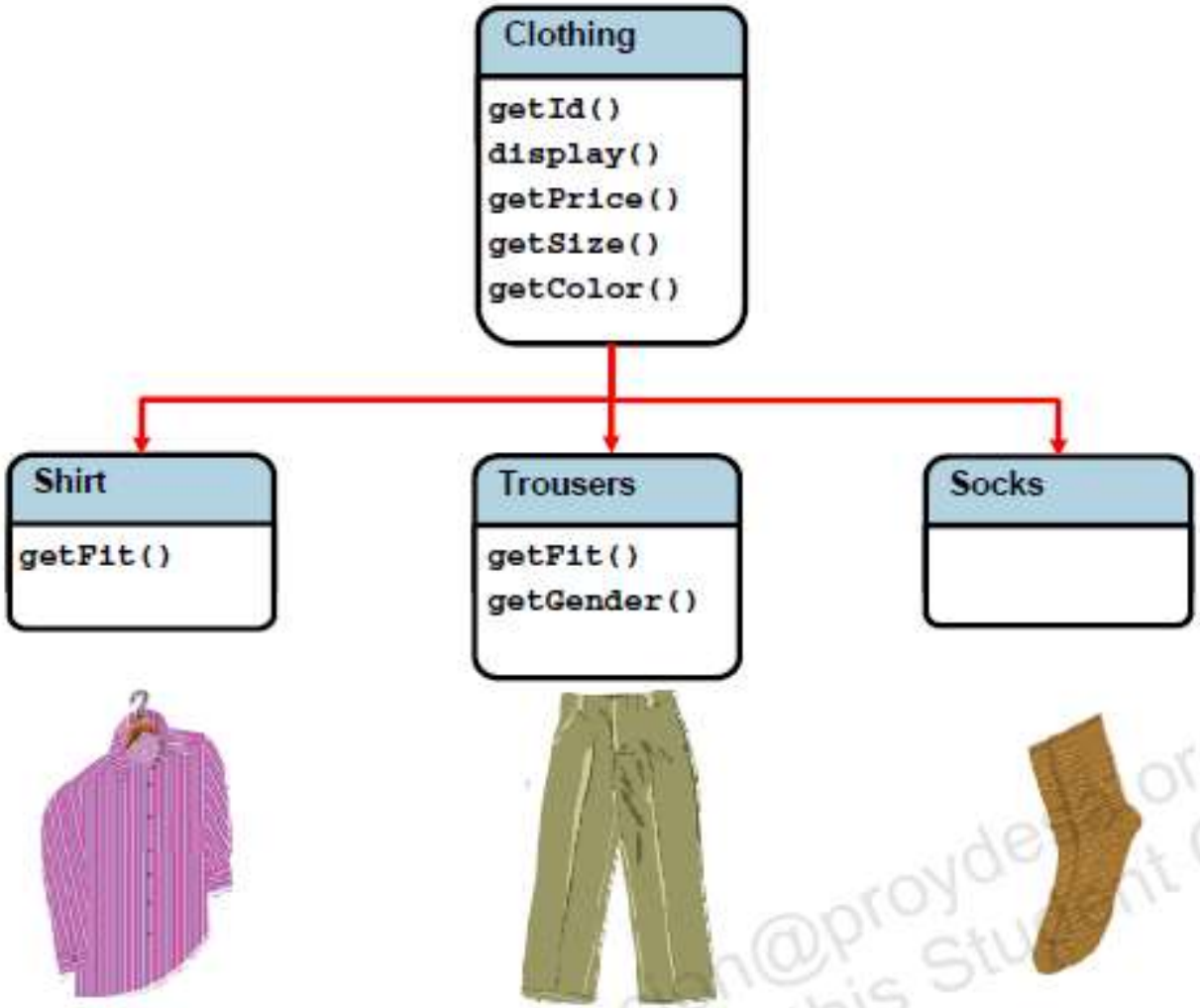
Comportamientos comunes

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

Duplicación de código



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 {

  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);

     this.fit = fit;
  }

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

Se asegura de que Shirt hereda los miembros de Clothing.

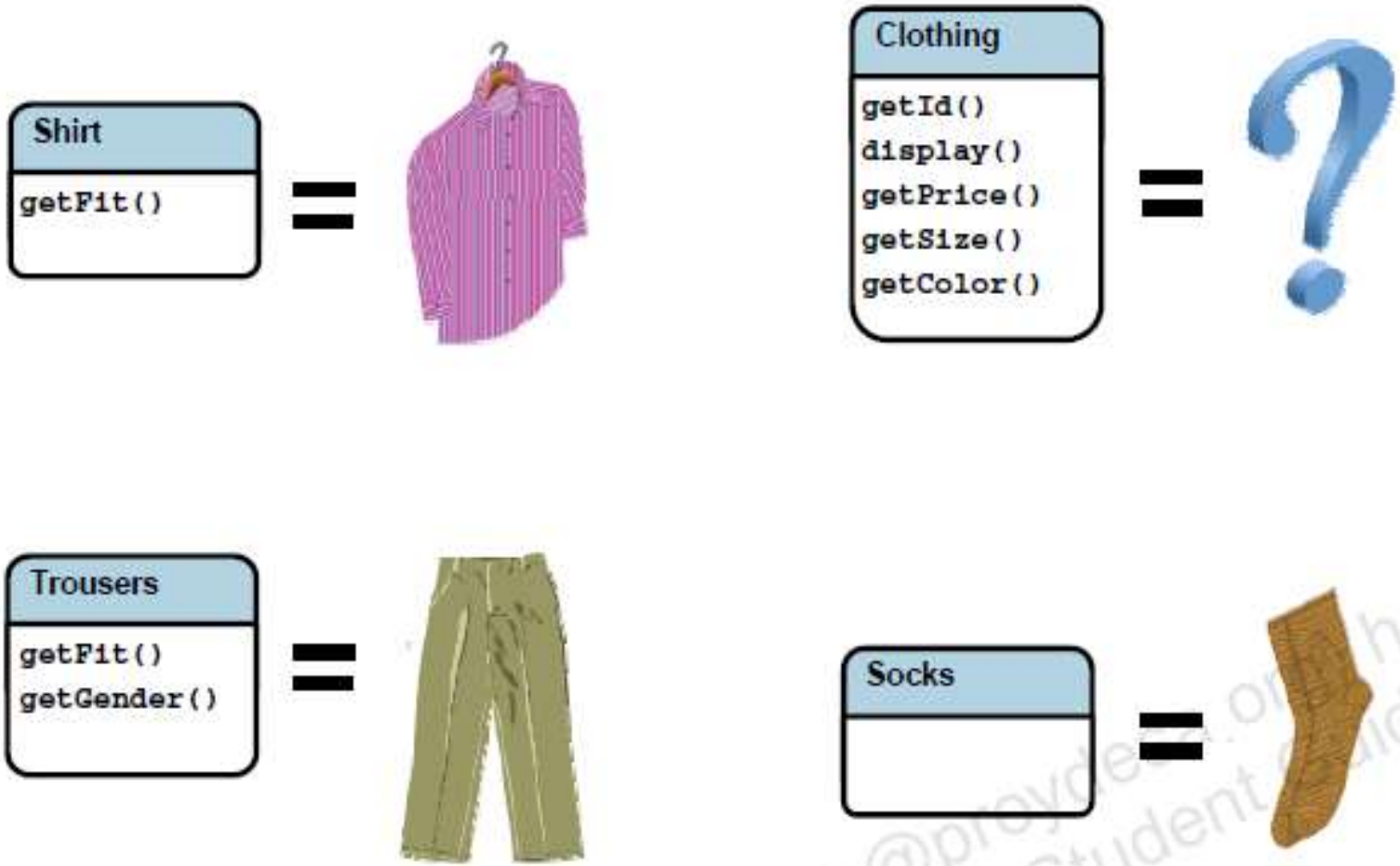
`super` es una referencia a los métodos y atributos de la superclase.

`this` es una referencia a este objeto.

Declaración de una sublcase: 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;
  }
}
```

Clases 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'; //
  private double price = 0.0; //  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 ...

}
```

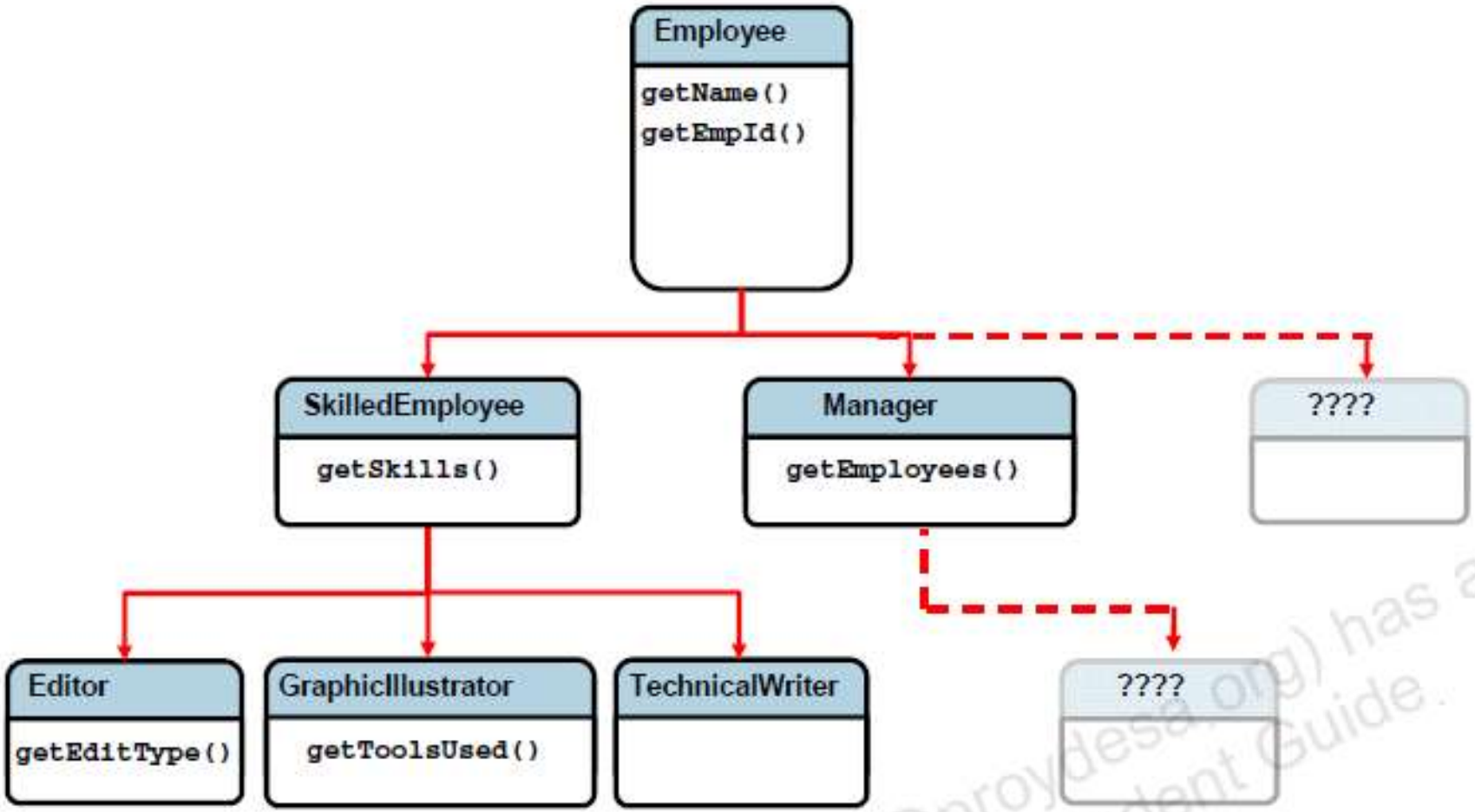
La palabra clave `abstract` asegura que estos valores se deben sustituir en la subclase.

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:

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

- Sin embargo, también puede utilizar la superclase como la referencia:

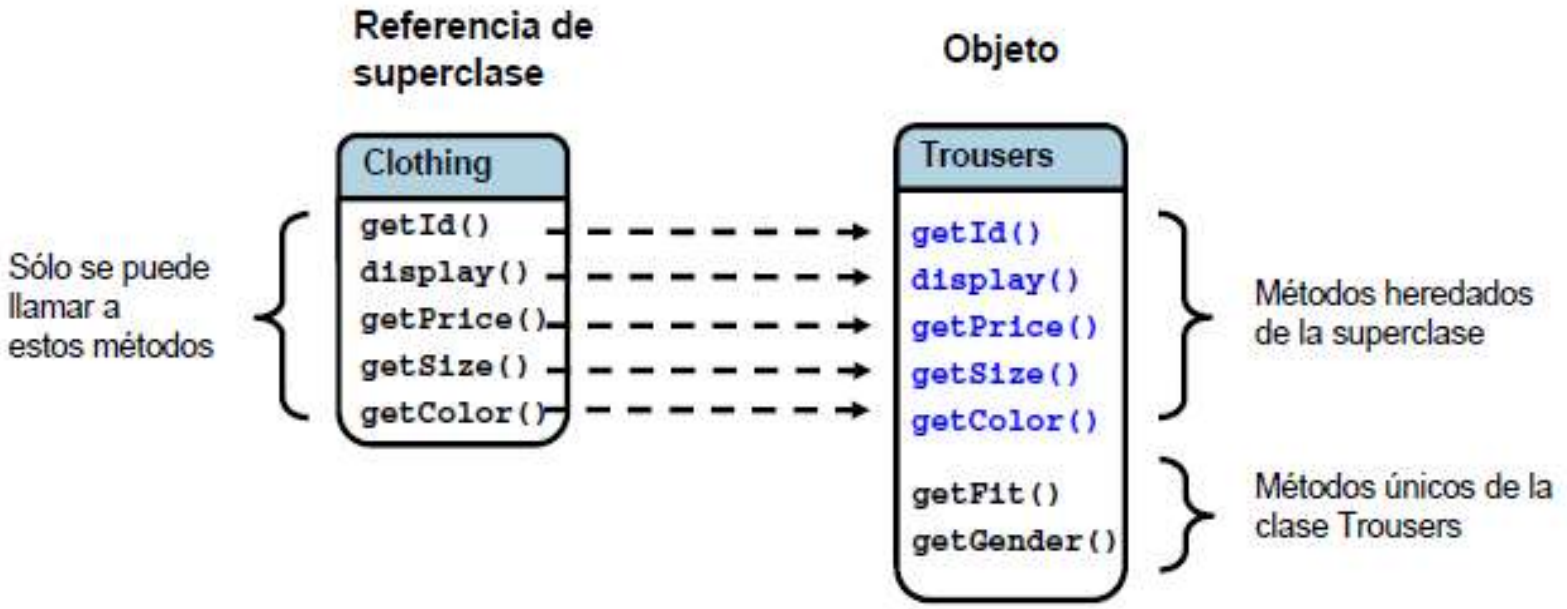
```
Clothing clothingItem1 = new Shirt();
Clothing clothingItem2 = new Trousers();
```

Acceso a funcionalidades de objeto

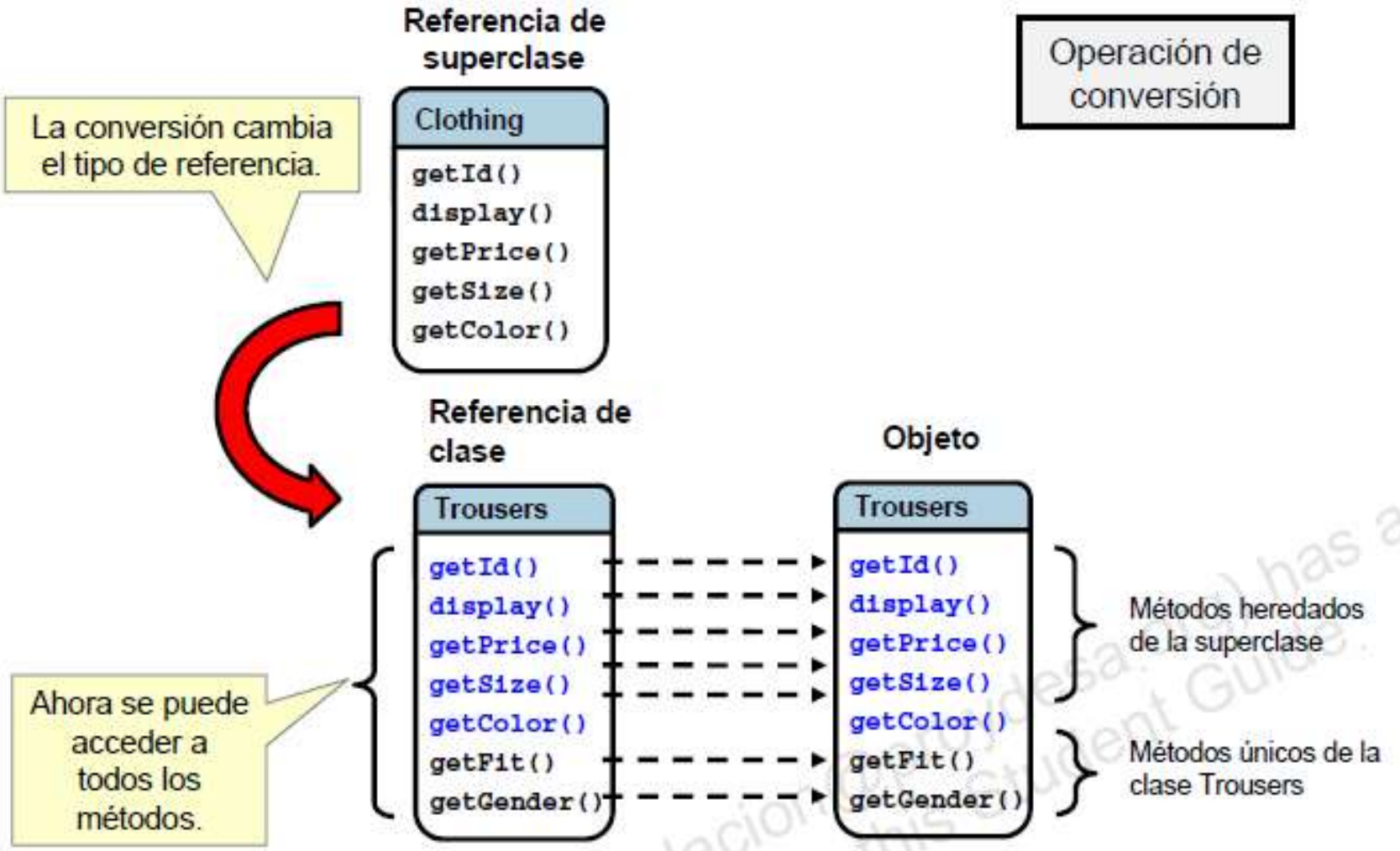
Juego completo de controles disponibles

Subjuego de controles disponibles

Acceso a métodos de clase desde la superclase



Conversión del tipo de referencia



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:

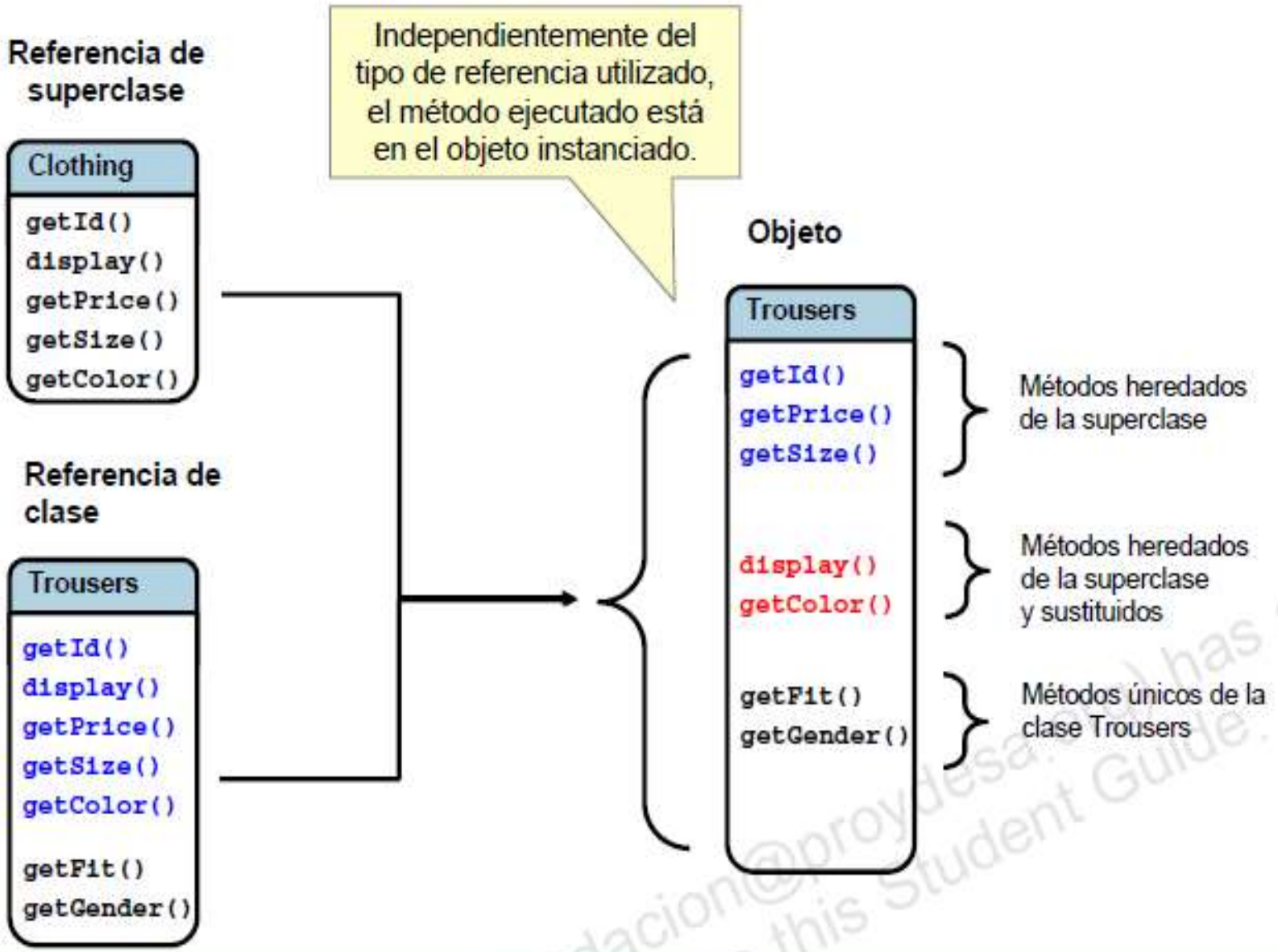
```
public static void displayDetails(Clothing cl) {

    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:

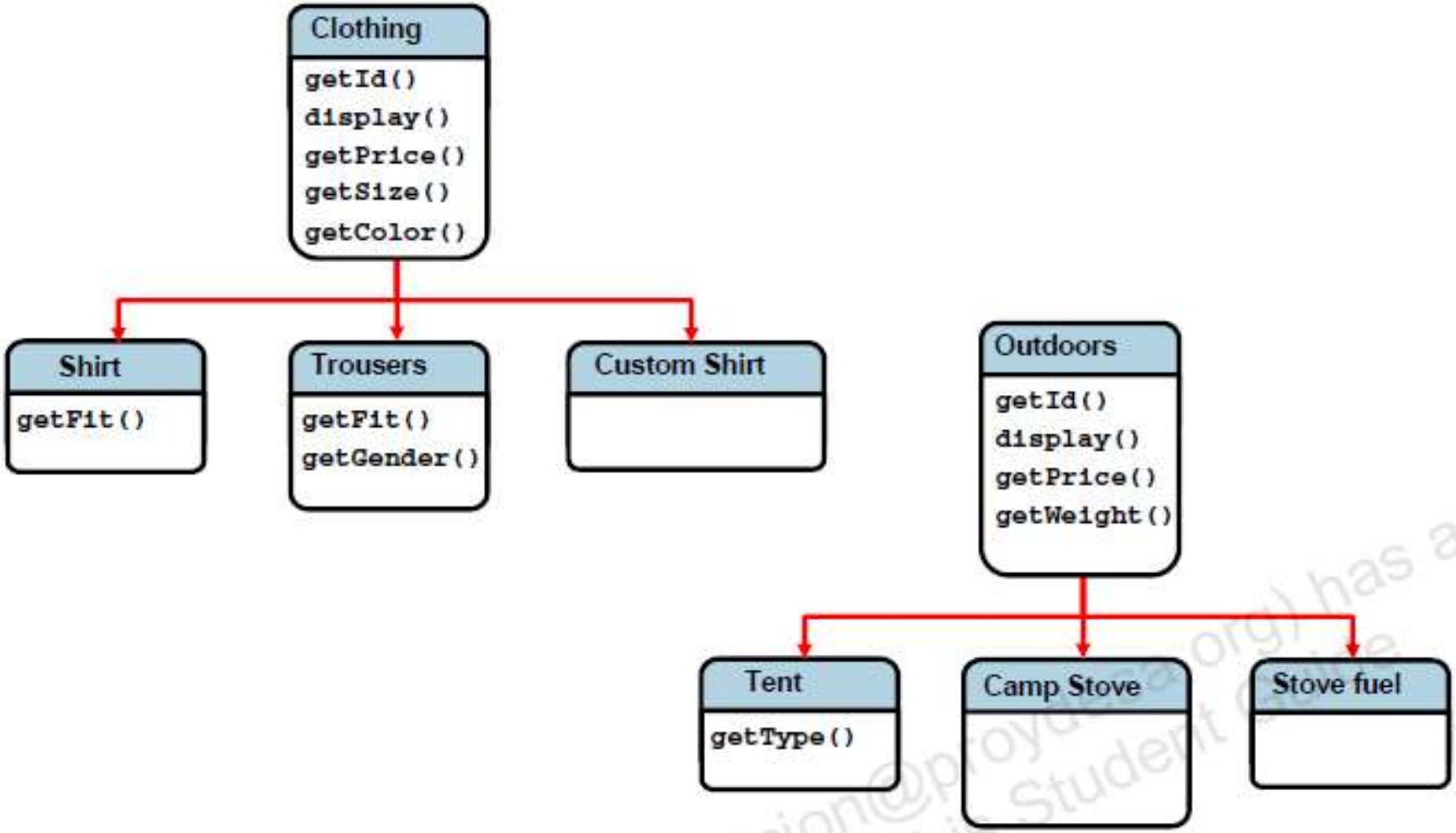
```
public static void displayDetails(Clothing cl) {
    cl.display();
    if (cl instanceof Trousers) {
        char fitCode = ((Trousers) cl).getFitCode();
        System.out.println("Fit: " + fitCode);
    }
    else { // Take some other action }
```

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

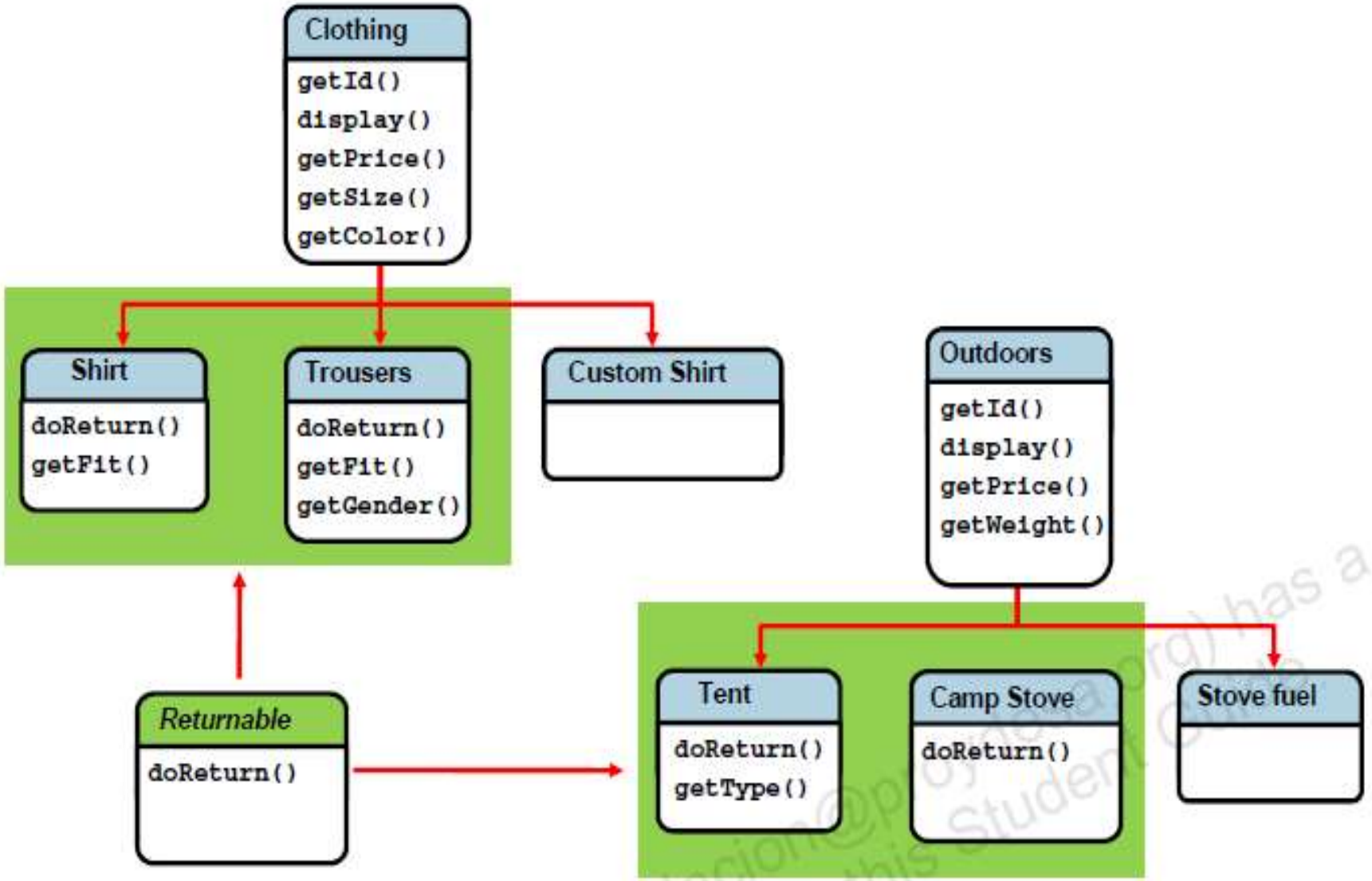
Llamadas a métodos polimórficos



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.

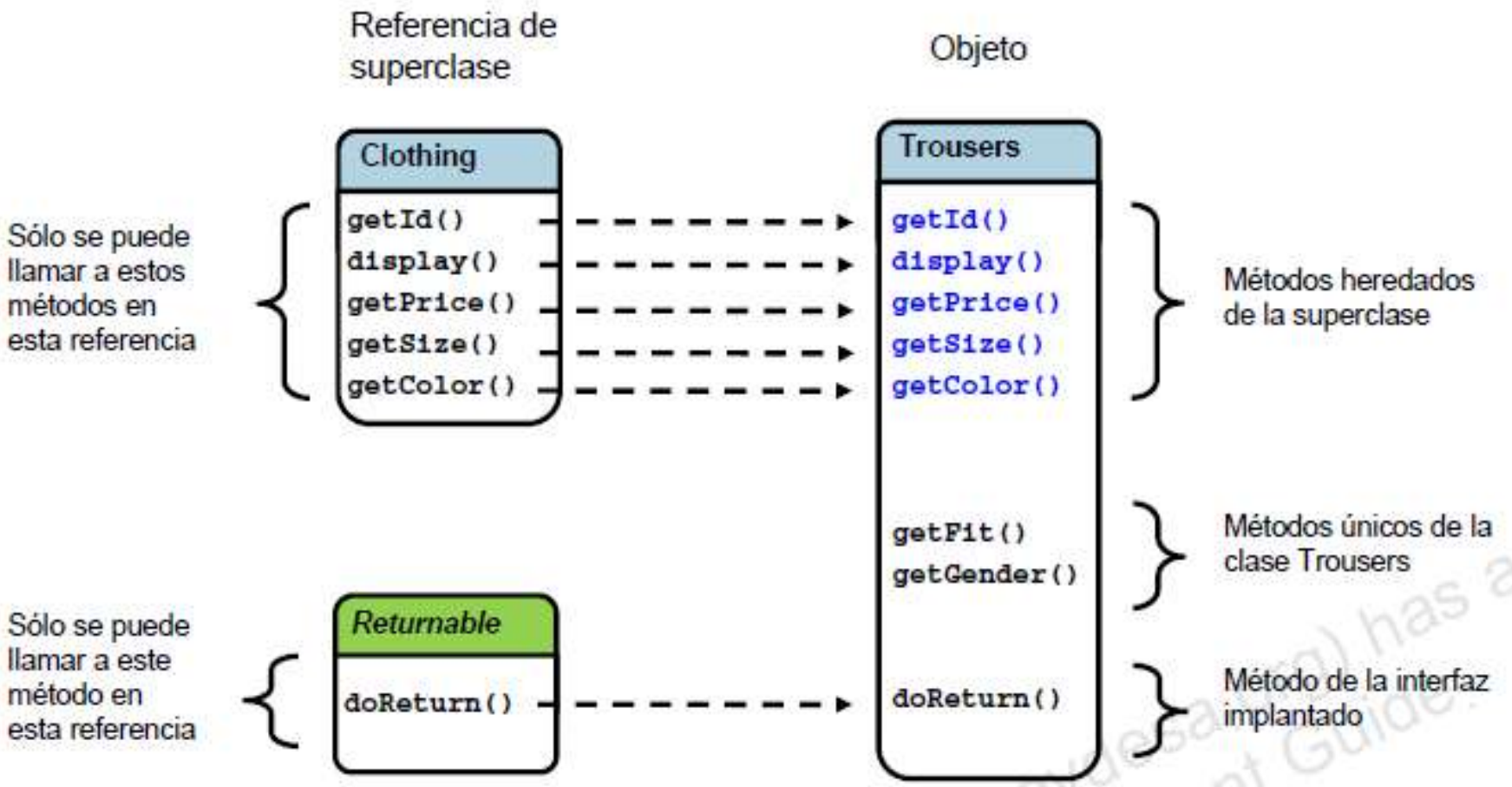
Asegura que Shirt debe implantar todos los métodos de Returnable.

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



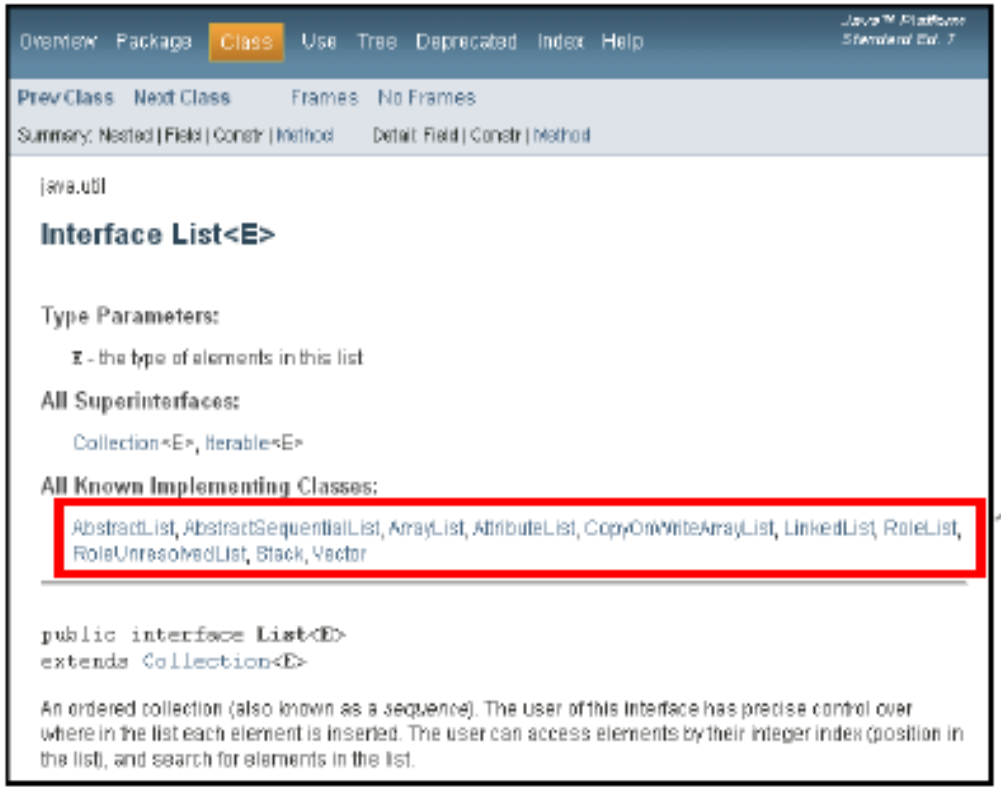
ArrayList

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

ArrayList implanta una serie de interfaces.

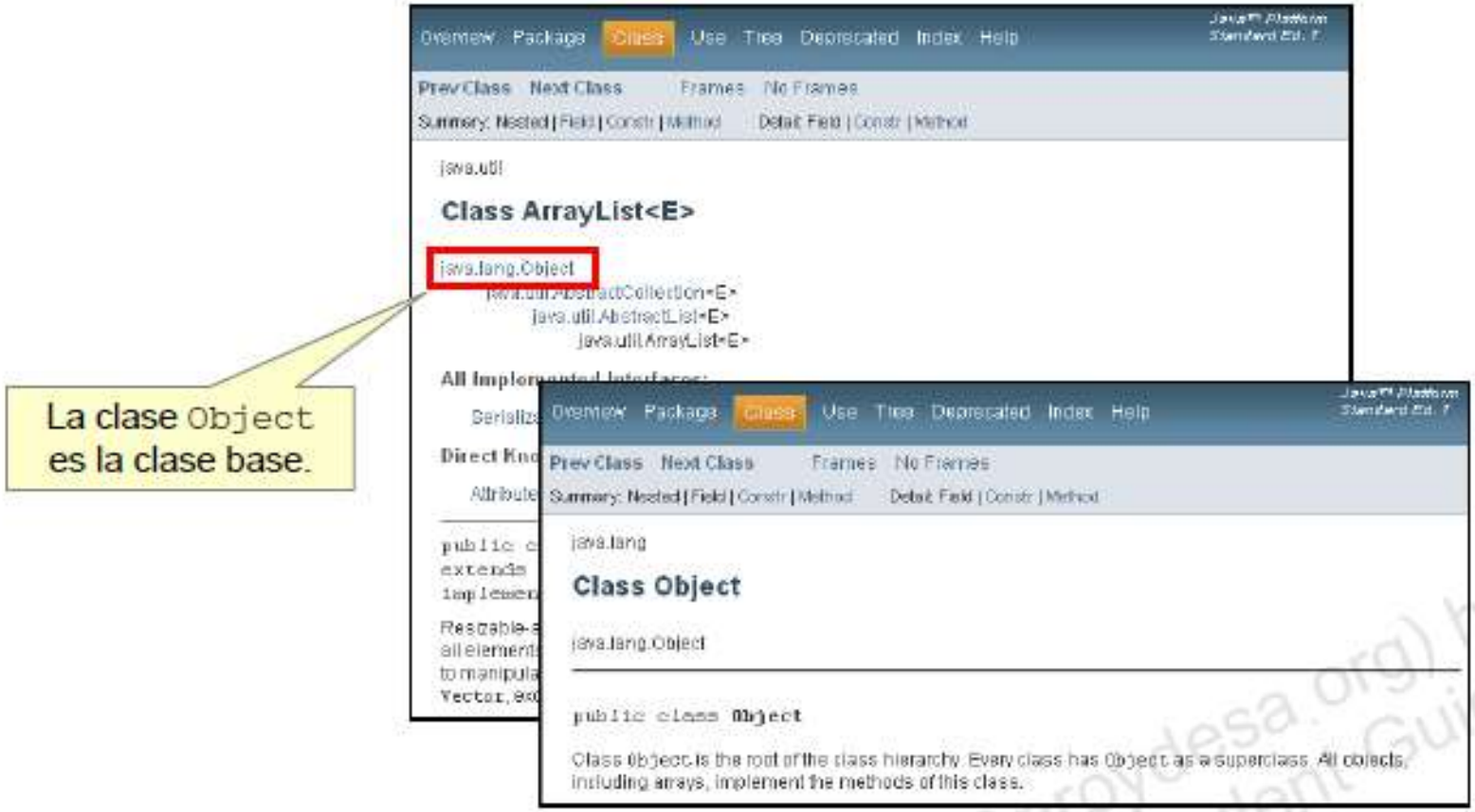
La interfaz List es la que se utiliza principalmente al trabajar con ArrayList.

Interfaz List



Existen varias clases que implantan la interfaz List

Clase `Object`



Llamada al método `toString()`

Se utiliza `toString()` de Object.

StringBuilder sustituye `toString()` de Object.

First hereda `toString()` de Object.

Second sustituye `toString()` de Object.

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