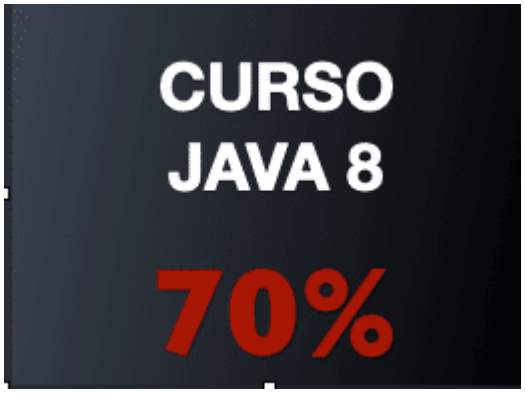
Las opciones soportadas por Java 8 Lambda Syntax siempre son costosas de aprender e integrar como parte de nuestro conocimiento Java. ¿Cuales son las sintaxis soportadas por las expresiones lamdba? .Vamos a construir un ejemplo sencillo usando un Java Comparator. Para ello nos vamos a construir la clase Persona y con nombre, apellidos y edad y vamos a crear un comparador por edad con Java clásico.




```
package com.arquitecturajava;

public class Persona {

    private String nombre;
    private String apellidos;
    private int edad;
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getApellidos() {
        return apellidos;
    }
```

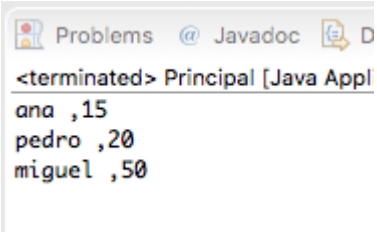
```
    public void setApellidos(String apellidos) {
        this.apellidos = apellidos;
    }
    public int getEdad() {
        return edad;
    }
    public void setEdad(int edad) {
        this.edad = edad;
    }
    public Persona(String nombre, String apellidos, int edad) {
        super();
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.edad = edad;
    }

}
```

```
package com.arquitecturajava;

import java.util.Comparator;

public class PersonaEdadComparator implements Comparator<Persona> {
        @Override
        public int compare(Persona p1, Persona p2) {
                return p1.getEdad()>p2.getEdad()? 1:-1;
        }

}
```

Una vez tenemos el comparador construido el siguiente paso es construir un programa main que nos ordene la lista:

```
package com.arquitecturajava;

import java.util.Arrays;
import java.util.List;

public class Principal {

    public static void main(String[] args) {

        Persona personaA = new Persona("pedro", "perez", 20);
        Persona personaB = new Persona("ana", "blanco", 15);
        Persona personaC = new Persona("miguel", "alvarez", 50);

        List < Persona > lista = Arrays.asList(personaA, personaB,
personaC);

        lista.sort(new PersonaEdadComparator());
        for (Persona p: lista) {

            System.out.println(p.getNombre() + " ," + p.getEdad());

        }

    }

}
```

Imprimimos el resultado:



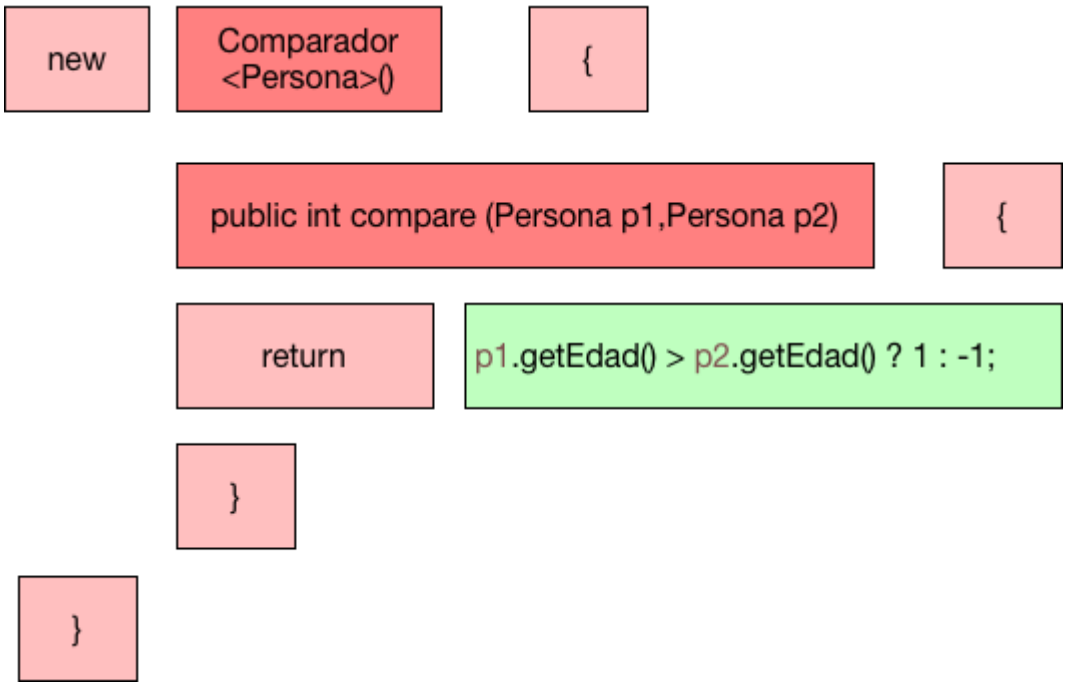
Java 8 Lambda Syntax vs InnerClasses.

Un muchos casos la gente suele preguntas si para ordenar la lista de elementos hace falta crear una nueva clase ya que es un poco pesado. La respuesta hasta Java 7 es que no es obligatorio y que es posible usar una clase anónima.

```
import java.util.Comparator;
import java.util.List;

public class Principal2 {

    public static void main(String[] args) {

        Persona personaA = new Persona("pedro", "perez", 20);
        Persona personaB = new Persona("ana", "blanco", 15);
        Persona personaC = new Persona("miguel", "alvarez", 50);

        List < Persona > lista = Arrays.asList(personaA, personaB,
personaC);
        lista.sort(new Comparator < Persona > () {
            @Override
            public int compare(Persona p1, Persona p2) {
                return p1.getEdad() > p2.getEdad() ? 1 : -1;
            }
        });
        for (Persona p: lista) {
            System.out.println(p.getNombre() + " ," + p.getEdad());
```

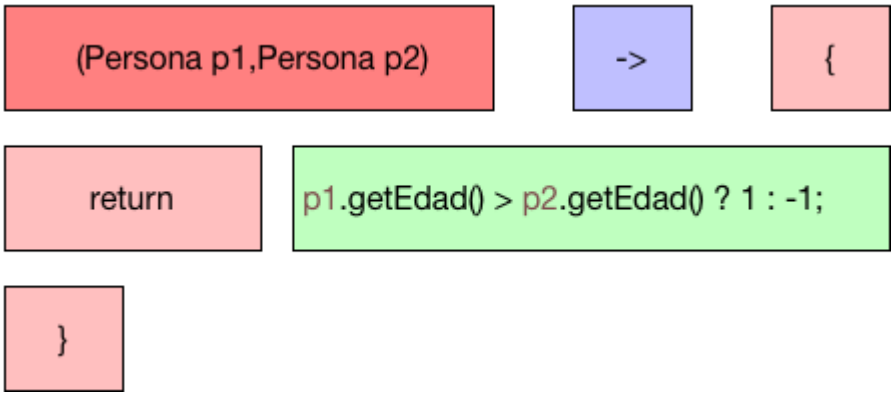
```
        }
    }
}
```

El resultado en la pantalla será el mismo , sin embargo ya no hará falta construir una clase PersonaEdadComparator . Es evidente que hay ventajas sin embargo también es evidente que la sintaxis es muy compleja y no ayuda a clarificar el código. Vamos a hacer un análisis de esta sintaxis.



Java 8 Lambda Syntax Block

La realidad es que tenemos mucho código que parece que aporta poco , sin embargo recordemos que si estamos todavía sobre el paraguas de Java 7 , la sintaxis es obligatoria. ¿Cómo podemos simplificarla con Java 8? . Podemos hacer uso de expresiones lambda y eliminar gran parte del código:



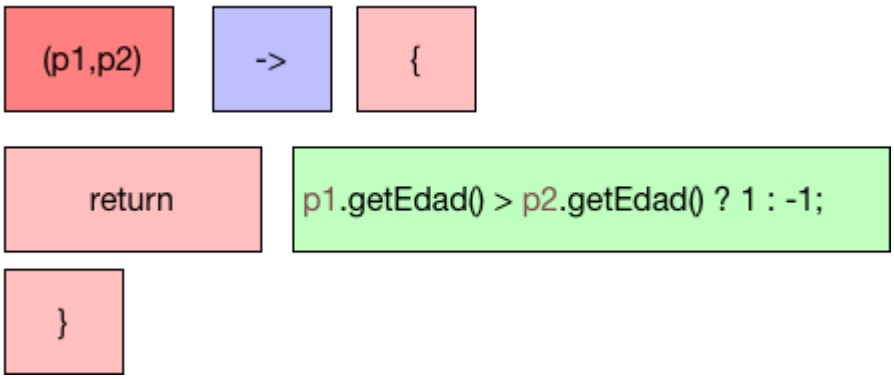
En este caso lo que hemos hecho es eliminar la clase anónima y quedarnos únicamente con lo que se denomina un bloque lambda el cual implementa la funcionalidad. Si mostramos el código las cosas quedan mucho más sencillas.

```
package com.arquitecturajava;

import java.util.Arrays;
import java.util.Comparator;
import java.util.List;

public class Principal3 {

    public static void main(String[] args) {

        Persona personaA = new Persona("pedro", "perez", 20);
        Persona personaB = new Persona("ana", "blanco", 15);
        Persona personaC = new Persona("miguel", "alvarez", 50);

        List < Persona > lista = Arrays.asList(personaA, personaB,
personaC);

        lista.sort((Persona p1, Persona p2) - > {
            return p1.getEdad() - > p2.getEdad() ? 1 : -1;
```

```
        });
        for (Persona p: lista) {

            System.out.println(p.getNombre() + " ," + p.getEdad());

        }

    }

}
```

Java 8 Lambda Expression

Es una simplificación importante . Ahora bien las expresiones lamdba soportan más posibilidades , una de ellas es lo que denomina inferred types es decir el compilador es capaz de entender de que tipo son las variables que el método sort necesita ya que estamos trabajando con una lista genérica de Personas. Así pues podemos simplificar todavía un poco más.

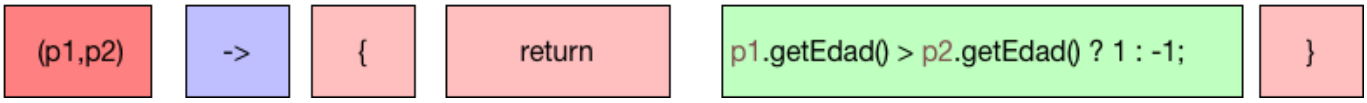


Veamos el código:

```
package com.arquitecturajava;

import java.util.Arrays;
import java.util.Comparator;
```

```
import java.util.List;

public class Principal4 {

	public static void main(String[] args) {

		Persona personaA = new Persona("pedro", "perez", 20);
		Persona personaB = new Persona("ana", "blanco", 15);
		Persona personaC = new Persona("miguel", "alvarez",
50);

		List<Persona> lista = Arrays.asList(personaA,
personaB, personaC);

		lista.sort(( p1, p2)->; {return p1.getEdad() ->
p2.getEdad() ? 1 : -1;});
		for (Persona p : lista) {

			System.out.println(p.getNombre() + " ," +
p.getEdad());

		}

	}

}
```

Hemos optimizado el código bastante , pero todavia no es suficiente , estamos ante lo que se denomina un lambda block , existe la posibilidad de compactar más el código y pasar a una expresión lamdba que únicamente ocupe una linea.



Esta sintaxis ya más directa veamosla en código:

```
import java.util.Arrays;
import java.util.Comparator;
import java.util.List;

public class Principal4 {

	public static void main(String[] args) {

		Persona personaA = new Persona("pedro", "perez", 20);
		Persona personaB = new Persona("ana", "blanco", 15);
		Persona personaC = new Persona("miguel", "alvarez",
50);

		List<Persona> lista = Arrays.asList(personaA,
personaB, personaC);

		lista.sort(( p1, p2)-> {return p1.getEdad() ->
p2.getEdad() ? 1 : -1;});
		for (Persona p : lista) {

			System.out.println(p.getNombre() + " ," +
p.getEdad());

		}

	}
```

```
}
```

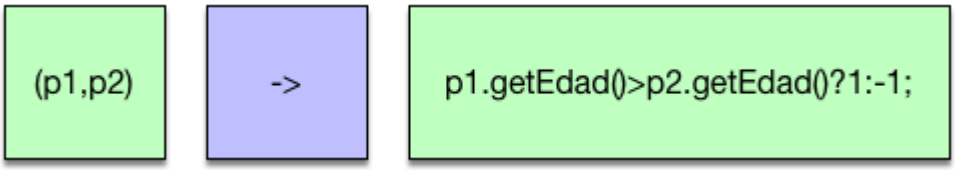
Cursos Relacionados Oferta 70%

- Curso de Spring Boot

- Curso de Spring WebFlux

- Curso de Java 8

Podemos dar un paso adicional más ya que el compilador es capaz de entender que al ser en una única linea se pueden eliminar los paréntesis y el return.



Ahora nos hemos quedado con la sintaxis más compacta.

```
lista.sort(( p1, p2)->p1.getEdad() > p2.getEdad() ? 1 : -1);
```

El uso de Java 8 lambda Syntax , es cada día más importante para conseguir un código más

compacto y mas sencillo de entender.Tenemos poco a poco que ir eliminando el uso de clases anónimas de nuestro código.

Otros artículos relacionados:

1. Java 8 Lambda Expressions (I)
2. Java Lambda reduce y wrappers
3. Java 8 Lambda y forEach (II)

Links Externos:

Expresiones Lambda