

Expresiones lambda

Definición

- **Expresión lambda.** Es una representación de una función anónima.
- Una expresión lambda puede ser asignada a una variable o pasada como argumento a un método.
- Las expresiones lambda no tienen nombre, pero tienen una lista de parámetros, cuerpo, tipo de retorno y posiblemente una lista de excepciones que puede lanzar.
- Sintaxis:

(parámetros) -> expresión **// lambda estilo expresión**

(parámetros) -> { instrucciones } **// lambda estilo bloque**

Definición

- En resumen, una expresión lambda es un método sin nombre.
- Ejemplos:
 $(\text{int } x, \text{int } y) \rightarrow \{ \text{return } x + y; \}$
 $x \rightarrow x * x$
 $() \rightarrow x$
- Una expresión lambda puede tener cero o más parámetros separados por comas.
- El tipo de los parámetros puede declararse o puede inferirse explícitamente del contexto.



Definición

- No se necesitan paréntesis alrededor de un solo parámetro.
- () se utiliza para indicar cero parámetros.
- El cuerpo puede contener cero o más instrucciones.
- No se necesitan llaves si el cuerpo tiene una sola instrucción.



Beneficios de las expresiones lambda

- Permite escribir código más compacto.
- Facilita la programación paralela.
- Permite pasar funciones como parámetros a métodos.

Ejemplo 1

- Suponer que se tiene un `ArrayList` de personas:
`List<Person> people = new ArrayList<>();`
`people.add(...);`
`...`
`people.add(...);`
- Suponer que se desea imprimir cada persona de la lista.
- Una forma es usar un `foreach`:

Ejemplo 1

// Opción 1 – foreach

```
for (Person p : people) {  
    System.out.println(p);  
}
```

Ejemplo 1

- Los objetos `ArrayList` tienen un método `forEach`:

```
void      forEach(Consumer<? super E> action)
```

Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception.

- Esto significa que se puede invocar:
 `people.forEach(objeto-de-tipo-Consumer);`
- Y que se puede imprimir el arreglo con una instrucción.
- Se necesita crear un objeto de tipo `Consumer`.

Ejemplo 1

- Consumer es una interface con los siguientes métodos:

Modifier and Type	Method	Description
void	accept (T t)	Performs this operation on the given argument.
default Consumer <T>	andThen (Consumer <? super T> after)	Returns a composed Consumer that performs, in sequence, this operation followed by the after operation.

Ejemplo 1

// opción 2a – usar el método foreach con un objeto anónimo

```
people.forEach(new Consumer<Person>() {  
    @Override  
    public void accept(Person person)  
    {  
        System.out.println(person);  
    }  
});
```

Ejemplo 1

// opción 2b – usar el método foreach con una expresión lambda

```
people.forEach(p -> System.out.println(p));
```

- p es el parámetro de la expresión lambda.
- Cuando es un solo parámetro, no se requieren paréntesis.
- Los parámetros no se declaran si el compilador puede inferir sus tipos.
- System.out.println(p) es el cuerpo de la expresión lambda.
- Cuando es una sola instrucción, no se requieren corchetes.
- La flecha -> separa los parámetros del cuerpo.

Ejemplo 2

- Suponer que se desea ordenar la lista alfabéticamente por el nombre de la persona.
- Los objetos List tienen el método sort:

default void

sort(**Comparator**<? super **E**> c)

Sorts this list according to the order induced by the specified **Comparator**.

- Se necesita crear un objeto de tipo Comparator.

Ejemplo 2

- Comparator es una interface.
- La interface tiene varios métodos, pero solo uno es obligatorio de implementar:

int

compare(T o1, T o2)

Compares its two arguments for order.

Ejemplo 2

- Forma 1 – usar un objeto anónimo:

```
people.sort(new Comparator<Person>() {  
    @Override  
    public int compare(Person o1, Person o2)  
    {  
        return o1.getName().compareTo(o2.getName());  
    }  
});
```

Ejemplo 2

- Forma 2 – usar una expresión lambda:

```
people.sort((p1, p2) -> p1.getName().compareTo(p2.getName()));
```

- Esta instrucción ordena la lista de la A a la Z (orden natural).
- Para ordenar la lista de la Z a la A (orden inverso), se intercambia el orden de invocación:

```
people.sort((p1, p2) -> p2.getName().compareTo(p1.getName()));
```

Ejemplo 3

- Suponer que se desean eliminar datos de una lista de acuerdo a algún criterio en particular.
- Los objetos `ArrayList` tienen el método `removeIf`:

`boolean` `removeIf(Predicate<? super E> filter)`

Removes all of the elements of this collection that satisfy the given predicate.

Ejemplo 3

- Predicate es una interface funcional cuyo método funcional es test.

boolean

test(T t)

Evaluates this predicate on the given argument.

Ejemplo 3

// borra las personas mayores de 25 años

```
people.removeIf(p -> p.age() > 25);
```

¿Dónde se usan las expresiones lambda?

- Respuesta: con *interfaces funcionales*.
- **Interface funcional.** Es una interface que especifica exactamente un solo método abstracto.

```
public interface Adder {  
    int add(int a, int b);  
}
```

- Las expresiones lambda permiten implementar el método abstracto en una línea y *tratar la expresión completa como una instancia de la interface funcional*.



Interfaces funcionales en Java

- El paquete `java.util.function` contiene un listado de las interfaces funcionales.



Referencias a métodos

- Las **referencias a métodos** permiten utilizar las definiciones de métodos existentes y pasarlas como lambdas.

Ejemplo

// Usando lambdas

```
List<Integer> lista = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);  
lista.forEach(p -> System.out.println(p));
```

// Usando referencias a métodos

```
List<Integer> lista = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);  
lista.forEach(System.out::println);
```

Ventajas

- Las referencias a métodos permiten escribir menos código.
- Por ejemplo, ordenar una lista por distintos criterios:

```
people.sort(comparing(Person::getName)); // nombre A-Z
```

```
people.sort(comparing(Person::getName).reversed()); // nombre Z-A
```

```
people.sort(comparing(Person::getPhone)); // teléfono 0-9
```

```
people.sort(comparing(Person::getPhone).reversed()); // teléfono 9-0
```

```
// Ordena por edad y luego por nombre
```

```
people.sort(comparing(Person::getAge).
```

```
    thenComparing(Person::getName));
```