UNIVERSIDAD TECNOLÓGICA DEL PERÚ

Facultad de Ingeniería

Programación Orientada a Objetos

Sesión 13: Introducción a las Expresiones Lambda

Recordando:

¿Que vimos la clase pasada?

Universidad Tecnológica del Perú

Logro de aprendizaje

Al finalizar la sesión, el estudiante soluciona problemas aplicando expresiones lambda usando Java en la resolución de ejercicios.

Universidad Tecnológica del Perú

Utilidad

Las expresiones lambda en Java son una herramienta poderosa que no solo mejora la productividad y claridad del código, sino que también prepara a los estudiantes para trabajar en entornos de desarrollo modernos y avanzados. Dominar esta característica es esencial para cualquier estudiante de sistemas que desee construir una carrera sólida en el desarrollo de software.

Agenda

- Introducción a las Expresiones Lambda.
- Expresiones Lambda en Programación Orientada a Objetos.
- Uso de Expresiones Lambda como Argumento de Métodos

Universidad Tecnológica del Perú

Introducción a las Expresiones Lambda

Definición :

Las expresiones lambda, también conocidas como funciones lambda o funciones anónimas, son una característica de muchos lenguajes de programación que permite definir funciones pequeñas y anónimas in situ. Esto significa que puedes crear funciones sin asignarles un nombre y usarlas directamente en el lugar donde las necesites. Las expresiones lambda son especialmente útiles cuando necesitas una función simple para realizar una tarea específica, como aplicar una operación a una colección de datos.

Introducción a las Expresiones Lambda

Definición :

Por lo general, una expresión lambda consta de los siguientes elementos:

- **Parámetros:** Los valores de entrada que la función lambda acepta.
- **Operador Arrow (->):** Se utiliza para separar los parámetros del cuerpo de la función lambda.
- **Cuerpo de la Función:** El código que se ejecutará cuando se llame a la función lambda.

https://www.youtube.com/watch?v=E3BU2fvolfY

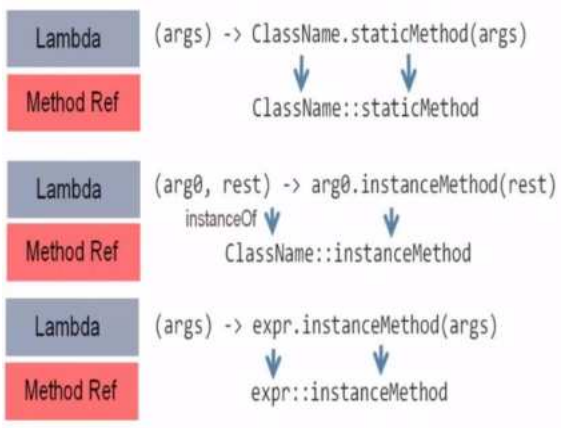
Universidad Tecnológica del Perú

Ventajas de Usar Expresiones Lambda en Programación

Las expresiones lambda proporcionan varias ventajas en programación:

Concisión del Código:

Las expresiones lambda permiten definir funciones pequeñas de manera concisa en una sola línea de código, lo que mejora la legibilidad y reduce la cantidad de código redundante.

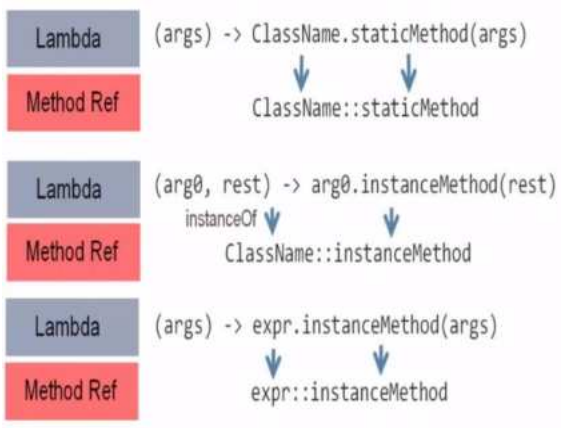


Universidad Tecnológica del Perú

Ventajas de Usar Expresiones Lambda en Programación

Programación Funcional: Facilitan la adopción de conceptos de programación funcional al permitir operaciones como el mapeo, filtrado y reducción de colecciones de datos de manera más intuitiva y elegante.

Mayor Abstracción: Puedes expresar la intención de tu código de manera más clara y abstracta al usar expresiones lambda.

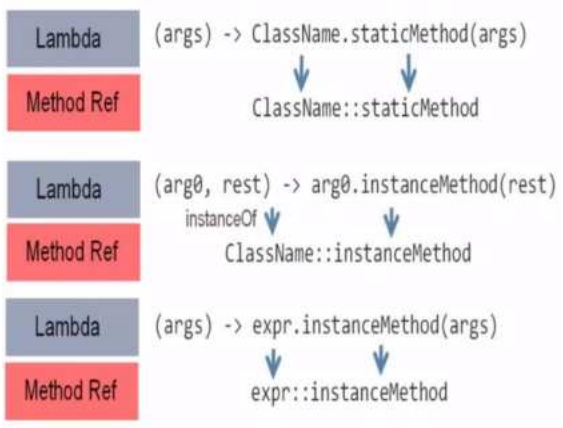


Universidad Tecnológica del Perú

Ventajas de Usar Expresiones Lambda en Programación

Mejora la Productividad: Al evitar la necesidad de definir funciones separadas con nombres, las expresiones lambda ahorran tiempo y aumentan la productividad.

Flexibilidad: Puedes utilizar expresiones lambda en una variedad de situaciones, como pasándolas como argumentos a funciones u ordenándolas.



Universidad Tecnológica del Perú

Expresiones Lambda en Programación Orientada a Objetos

Cómo se Aplican las Expresiones Lambda en un Entorno Orientado a Objetos:

En un contexto de Programación Orientada a Objetos (POO), las expresiones lambda se utilizan para definir funciones anónimas que pueden ser aplicadas a objetos y colecciones de objetos. A continuación, se detallan los aspectos clave de su aplicación:

Expresiones Lambda en Programación Orientada a Objetos

- **Métodos de Orden Superior:**
  Las expresiones lambda se usan en POO como ejemplos de métodos de orden superior, es decir, funciones que pueden recibir otras funciones como argumentos o devolver funciones como resultado. Esto permite realizar operaciones avanzadas en colecciones de objetos, como filtrado, mapeo y reducción, de manera más eficiente y legible.

Expresiones Lambda en Programación Orientada a Objetos

- **Iteración Mejorada:**
  Las expresiones lambda son especialmente útiles cuando necesitas iterar sobre una colección de objetos y aplicar una operación o función a cada elemento de manera sencilla.
  En lugar de escribir bucles explícitos, puedes usar funciones como map, filter, y reduce junto con expresiones lambda para lograr una iteración más elegante y eficiente.

Expresiones Lambda en Programación Orientada a Objetos

- **Mayor Abstracción:**

  Las expresiones lambda permiten abstraer las operaciones que se aplican a los objetos, lo que hace que el código sea más modular y mantenible.

  Esto significa que puedes cambiar la lógica de una operación sin necesidad de modificar todo el código que la utiliza.

ICODEAP

(Learn Code Java)

Expresiones Lambda en Java y Funciones Anónimas

Elivar Largo
Java Backend Developer

www.icodeap.com

https://www.youtube.com/watch?v=AeNiRg4aeVs&t=0s

Ejemplos de Uso de Expresiones Lambda en POO

- **Mapeo de una Lista:**
  Puedes usar una expresión lambda para mapear una lista de objetos a otra lista, aplicando una transformación a cada elemento. Por ejemplo, si tienes una lista de objetos Persona, puedes utilizar una expresión lambda para mapearla a una lista de sus nombres.

Ejemplos de Uso de Expresiones Lambda en POO

- **Filtrado de una Lista:**
  Las expresiones lambda son útiles para filtrar elementos de una lista según un criterio específico.
  Por ejemplo, puedes filtrar una lista de Producto para obtener solo los productos que tienen un precio superior a cierto valor.

Ejemplos de Uso de Expresiones Lambda en POO

- **Reducción de una Lista:**
  Puedes utilizar expresiones lambda en la reducción de una lista para calcular un valor agregado, como la suma de los valores de una lista de números.

Ejemplos de Uso de Expresiones Lambda en POO

- **Ordenamiento de una Lista:**
  Las expresiones lambda también se utilizan para definir un criterio de ordenamiento personalizado cuando necesitas ordenar una lista de objetos de acuerdo a un atributo específico.

Uso de Expresiones Lambda como Argumento de Métodos

En programación, es común que los métodos reciban argumentos, que son valores que se utilizan como entrada para realizar alguna acción dentro del método.

- Una de las características más potentes y versátiles de las expresiones lambda es la capacidad de ser pasadas como argumentos a otros métodos.
- Esto permite que las expresiones lambda se utilicen para personalizar y parametrizar el comportamiento de un método de manera flexible. A continuación, se describen los conceptos clave de cómo las expresiones lambda pueden ser pasadas como argumentos a métodos:

Uso de Expresiones Lambda como Argumento de Métodos:

Métodos de Orden Superior:

- En programación, los métodos que pueden aceptar funciones (expresiones lambda) como argumentos se denominan "métodos de orden superior."
- Estos métodos se pueden utilizar para aplicar una función específica en una variedad de contextos, lo que los hace muy flexibles y reutilizables.

Uso de Expresiones Lambda como Argumento de Métodos:

Interfaz Funcional:

- En muchos lenguajes de programación, como Java, se utilizan interfaces funcionales para definir la firma de métodos que aceptan expresiones lambda como argumentos.
- Estas interfaces funcionales contienen un solo método abstracto que debe ser implementado por la expresión lambda. Ejemplos de interfaces funcionales incluyen Predicate, Consumer, Function, entre otros.

Uso de Expresiones Lambda como Argumento de Métodos:

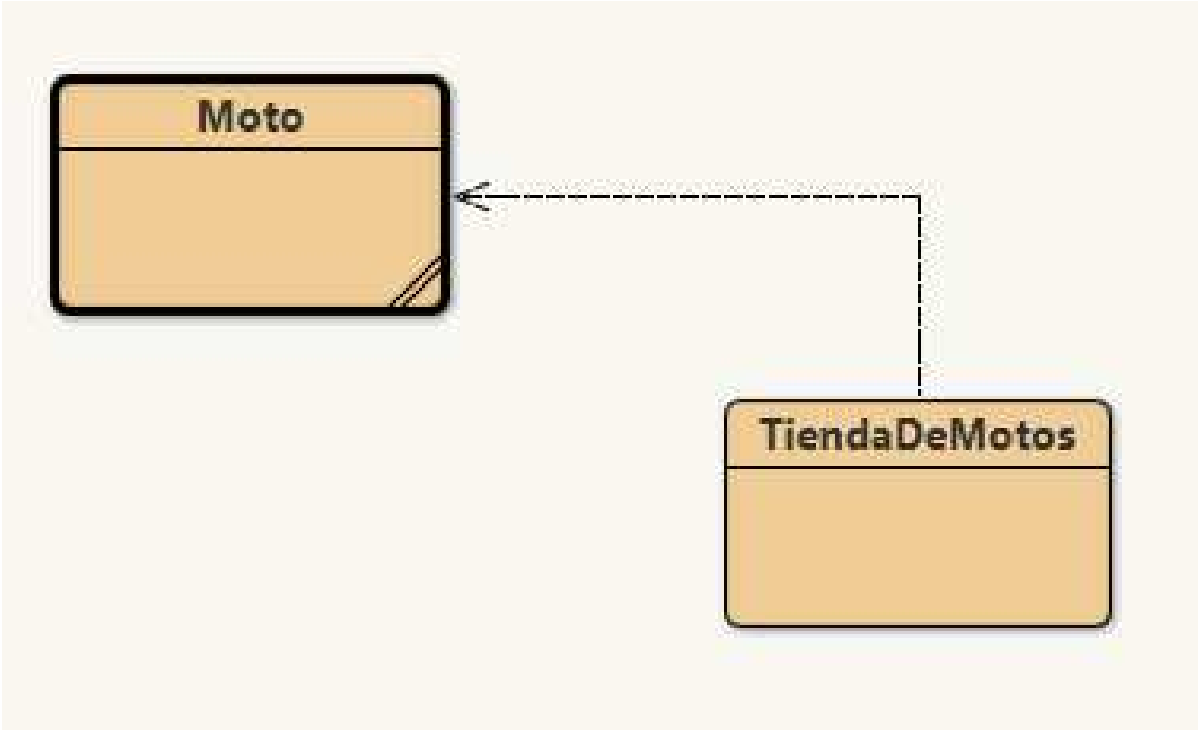
Personalización del Comportamiento:

- Pasar expresiones lambda como argumentos permite personalizar el comportamiento de un método sin tener que modificar su implementación.
- Esto es especialmente útil en situaciones donde deseas realizar operaciones específicas en una colección de datos o aplicar lógica particular a un proceso.

Ejercicio:

- Ejercicio simple utilizando un método de transformación en un entorno de desarrollo como NetBeans para una tienda de venta de motos.
- En este ejemplo, supongamos que tienes una lista de objetos que representan las motos en el inventario de la tienda y deseas transformar estos objetos en una lista de cadenas que contienen información sobre cada moto para mostrar en una página web o en una interfaz de usuario. Para ello, utilizaremos expresiones lambda y el método de transformación map.

Ejercicio:



```
public class Moto {
    private String marca;
    private String modelo;
    private int precio;

    // Constructor
    public Moto(String marca, String modelo, int precio) {
        this.marca = marca;
        this.modelo = modelo;
        this.precio = precio;
    }

    // Métodos getters
    public String getMarca() {
        return marca;
    }

    public String getModelo() {
        return modelo;
    }

    public int getPrecio() {
        return precio;
    }
```

Ejercicio: class Moto

Ejercicio: class Moto

```
// Métodos setters (opcional, solo si necesitas cambiar los valores después de la creación)
    public void setMarca(String marca) {
        this.marca = marca;
    }

    public void setModelo(String modelo) {
        this.modelo = modelo;
    }

    public void setPrecio(int precio) {
        this.precio = precio;
    }
}
```

Ejercicio: class TiendaDe Motos

Vamos a usar un método de transformación map para convertir una lista de objetos Moto en una lista de cadenas que contienen información sobre cada moto.

```
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
public class TiendaDeMotos {
    public static void main(String[] args) {
        // Crear una lista de motos
        List<Moto> inventario = new ArrayList<>();
        inventario.add(new Moto("Honda", "CBR500R", 6500));
        inventario.add(new Moto("Kawasaki", "Ninja 300", 4500));
        inventario.add(new Moto("Suzuki", "GSX-R750", 9500));
```

Ejercicio: class TiendaDe Motos

```
// Usar una expresión lambda y el método map para transformar las motos en cadenas
        List<String> informacionDeMotos = inventario.stream()
            .map(moto -> "Marca: " + moto.getMarca() + ", Modelo: " + moto.getModelo() + ",
Precio: $" + moto.getPrecio())
            .collect(Collectors.toList());

        // Imprimir la información de las motos transformadas
        for (String infoMoto : informacionDeMotos) {
            System.out.println(infoMoto);
        }
    }
}
```

Explicación del Ejercicio

Este código forma parte de una operación utilizando streams en Java, específicamente sobre una lista (o flujo) de objetos `Moto`. Vamos a desglosar lo que hace cada parte:

1. `.map(moto -> "Marca: " + moto.getMarca() + ", Modelo: " + moto.getModelo() + ", Precio: $" + moto.getPrecio())`:

   - `.map()` es una operación de stream que transforma cada elemento del stream según la función proporcionada.
   - `moto -> ...` es una expresión lambda que toma cada objeto `moto` del stream y realiza una operación sobre él.
   - `"Marca: " + moto.getMarca() + ", Modelo: " + moto.getModelo() + ", Precio: $" + moto.getPrecio()` es la operación que se aplica a cada objeto `moto`. Concatena las propiedades (`marca`, `modelo` y `precio`) de la moto en una cadena formateada que representa la información de la moto.

Explicación del Ejercicio

2. `.collect(Collectors.toList())`:

    - `.collect()` es otra operación de stream que recolecta los elementos del stream en una colección.
    - `Collectors.toList()` es un colector que recolecta los elementos del stream en una lista.

Por lo tanto, en resumen:

- El código toma una lista de objetos `Moto`.
- Con la operación `.map()`, cada objeto `Moto` se transforma en una cadena de texto que contiene información específica (`Marca: ...`, `Modelo: ...`, `Precio: ...`).
- La operación `.collect(Collectors.toList())` recolecta estas cadenas de texto transformadas en una lista de cadenas (`List<String>`).

Explicación del Ejercicio

En este ejemplo, hemos creado una lista de objetos Moto en el inventario de la tienda y luego utilizamos una expresión lambda y el método map para transformar cada objeto Moto en una cadena que contiene información sobre la marca, modelo y precio de la moto.
Finalmente, recopilamos las cadenas transformadas en una lista de cadenas y las imprimimos en la consola.

Explicación del Ejercicio

Así, al final de esta operación, informacionDeMotos será una lista que contiene cadenas formateadas con la información de cada objeto Moto en el inventario. Cada cadena representará algo similar a "Marca: [marca], Modelo: [modelo], Precio: $[precio]".

Practica

Implementa el ejercicio visto.

Conclusiones:

- Hemos explorado cómo las expresiones lambda pueden mejorar la programación orientada a objetos y cómo se pueden utilizar como argumentos de métodos.
- Al comprender estos conceptos, los programadores podrán escribir un código más limpio y eficiente, lo que les permitirá desarrollar aplicaciones de mayor calidad.
- Las expresiones lambda son una herramienta poderosa que puede mejorar significativamente la productividad y la capacidad de mantenimiento del código en proyectos de software.

Cierre

¿Qué hemos aprendido hoy?

Bibliografía

- MORENO PÉREZ, J. “Programación orientada a objetos”. RA-MA Editorial. https://tubiblioteca.utp.edu.pe/cgi-bin/koha/opac-detail.pl?biblionumber=31933
- Vélez Serrano, José. “Diseñar y programar, todo es empezar: una introducción a la Programación Orientada a Objetos usando UML y Java”. Dykinson. https://tubiblioteca.utp.edu.pe/cgi-bin/koha/opac-detail.pl?biblionumber=36368

Universidad Tecnológica del Perú