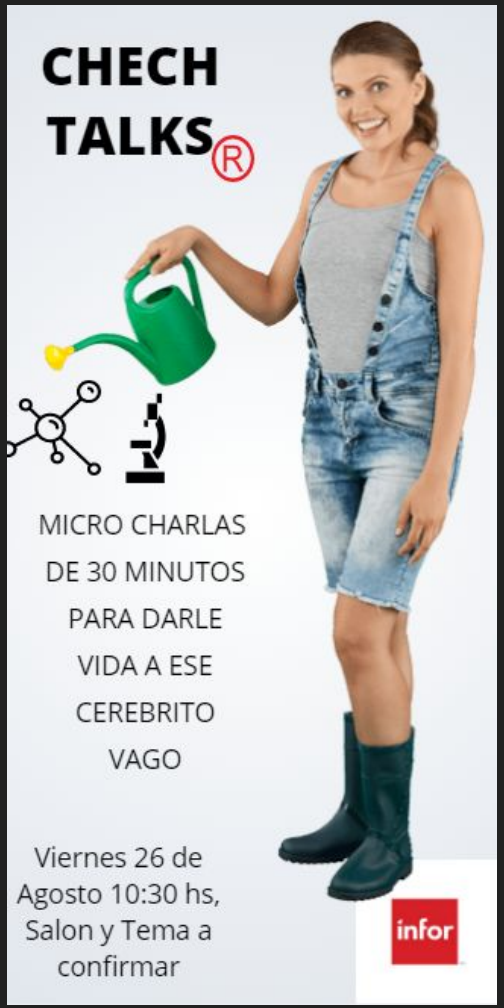
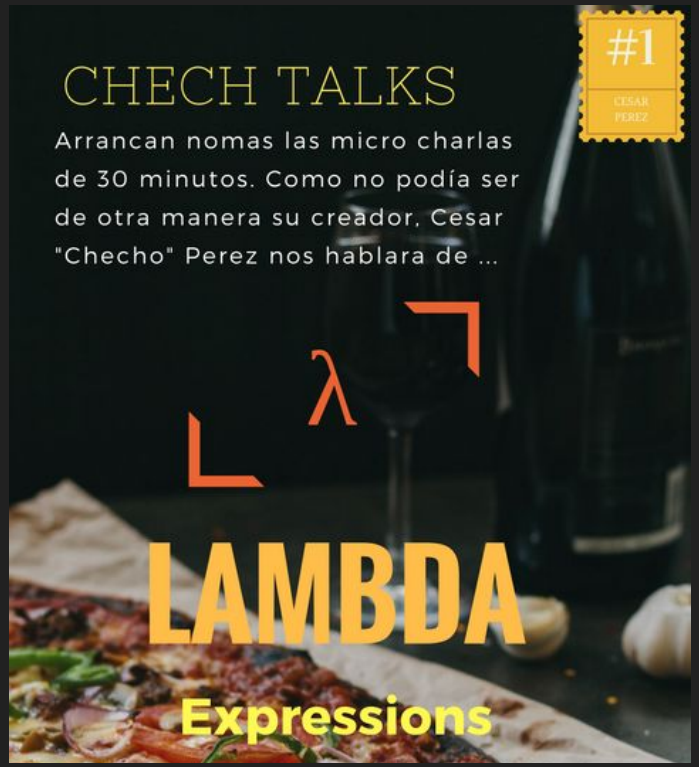


- ➔ Menos de 1 hora para prepararla
- ➔ El disertante puede proponer temas, o el público sugerirlos.
- ➔ No hace falta ser un experto.
- ➔ Tema libre. La única regla es que le interese al público.
- ➔ Son opcionales, pero si no querés hablar sos alto topu.

chech-talk I


CHECH TALKS
#1
CESAR PEREZ
Arrancan nomas las micro charlas de 30 minutos. Como no podía ser de otra manera su creador, Cesar "Checho" Perez nos hablara de ...
λ
LAMBDA
Expressions

Índice

- Definición
- Java 8
  - Uso
  - Functional interface
  - Ejemplo en Enact
  - Performance
  - Best practices
- Links de interés

Cálculo lambda

From Wikipedia, the free encyclopedia

El cálculo lambda es un sistema formal diseñado para investigar la definición de función, la noción de aplicación de funciones y la recursión. Fue introducido por Alonzo Church y Stephen Kleene en la década de 1930; Church usó el cálculo lambda en 1936 para resolver el Entscheidungsproblem. **Puede ser usado para definir de manera limpia y precisa qué es una "función computable".** El interrogante de si dos expresiones del cálculo lambda son equivalentes no puede ser resuelto por un algoritmo general. Esta fue la primera pregunta, incluso antes que el problema de la parada, para el cual la indecidibilidad fue probada. **El cálculo lambda tiene una gran influencia sobre los lenguajes funcionales, como Lisp, ML y Haskell.**

Se puede considerar al cálculo lambda como el más pequeño lenguaje universal de programación. Consiste en una regla de transformación simple (sustitución de variables) y un esquema simple para definir funciones.

El cálculo lambda es universal porque cualquier función computable puede ser expresada y evaluada a través de él. Por lo tanto, es equivalente a las máquinas de Turing. Sin embargo, el cálculo lambda no hace énfasis en el uso de reglas de transformación y no considera las máquinas reales que pueden implementarlo. Se trata de una propuesta más cercana al software que al hardware.

Índice

- Definición
- Java 8
  - Uso
  - Functional interface
  - Ejemplo en Enact
  - Performance
  - Best practices
- Links de interés

Java 8

Lambda expressions provide a clear and concise way of implementing a single-method interface using an expression. It allows to reduce the amount of code you have to create and maintain. It's often a concise replacement for anonymous classes.

Lambdas can only operate on a **functional interface**, which is an interface with just one abstract method. Functional interfaces can have any number of default or static methods.

Java 8

Con una clase anónima

```
Collections.sort(
    personList,
    new Comparator<Person>() {
        public int compare(Person p1, Person p2){
            return p1.getFirstName().compareTo(p2.getFirstName());
        }
    }
);
```

Con una expresión lambda

```
Collections.sort(
    personList,
    (p1, p2) -> p1.getFirstName().compareTo(p2.getFirstName())
);
```

Java 8

Variantes

```
Collections.sort(
    personList,
    (p1, p2) -> p1.getFirstName().compareTo(p2.getFirstName())
);

Collections.sort(
    personList,
    (Person p1, Person p2) -> p1.getFirstName().compareTo(p2.getFirstName())
);

Collections.sort(
    personList,
    (Person p1, Person p2) -> {
        return p1.getFirstName().compareTo(p2.getFirstName())
    }
);
```

Índice

- Definición
- Java 8
  - Uso
  - Functional interface
  - Ejemplo en Enact
  - Performance
  - Best practices
- Links de interés

Functional Interface

```
@FunctionalInterface
public interface Comparator<T> {

    int compare(T o1, T o2);

    /* también tiene métodos default y static */
}
```

Índice

- Definición
- Java 8
  - Uso
  - Functional interface
  - Ejemplo en Enact
  - Performance
  - Best practices
- Links de interés

Ejemplo en Enact

http://git.code.us.starmount.internal/projects/RCS/repos/enact/commits/659696e7b4a388a3246f53ffa687c173e05f874c

Lambda expressions

```
public void getPasswordHistoricalNumber() {
        asyncTaskFactory
            .buildTask(PasswordService.class, new Caller<PasswordService, Integer>() {
                @Override
                public Integer call(PasswordService service) {
                    return service.getPasswordHistoricalNumber();
                }
            })
            .call()
            .done(new VaadinDoneCallback<Integer>() {
                @Override
                public void onDoneUI(final Integer result) {
                    String newValue =
                        resetPasswordListOfValidations.getValue().replace("X", String.valueOf(result));
                    resetPasswordListOfValidations.setValue(newValue);
                }
            })
            .fail(new VaadinFailCallback<Throwable>() {
                @Override
                public void onFailUI(Throwable result) {
                }
            });
    }
```

Abstract class to Functional Interface

```
public abstract class VaadinDoneCallback<T> implements DoneCallback<T> {
    protected final UI ui = UI.getCurrent();
    @Override
    public void onDone(final T result) {
        ui.access(new Runnable() {
            @Override
            public void run() {
                onDoneUI(result);
            }
        });
    }
    abstract public void onDoneUI(T result);
}
```

Abstract class to Functional Interface

```
@FunctionalInterface
public interface VaadinDoneCallback<T> extends DoneCallback<T> {
    UI ui = UI.getCurrent();
    @Override
    default void onDone(final T result) {
        ui.access(() -> onDoneUI(result));
    }
    void onDoneUI(T result);
}
```

Lambda expressions

```
public void getPasswordHistoricalNumber() {
        asyncTaskFactory
            .buildTask(PasswordService.class, new Caller<PasswordService, Integer>() {
                @Override
                public Integer call(PasswordService service) {
                    return service.getPasswordHistoricalNumber();
                }
            })
            .call()
            .done(new VaadinDoneCallback<Integer>() {
                @Override
                public void onDoneUI(final Integer result) {
                    String newValue =
                        resetPasswordListOfValidations.getValue().replace("X", String.valueOf(result));
                    resetPasswordListOfValidations.setValue(newValue);
                }
            })
            .fail(new VaadinFailCallback<Throwable>() {
                @Override
                public void onFailUI(Throwable result) {
                }
            });
    }
```

Lambda expressions

```
public void getPasswordHistoricalNumber() {
        asyncTaskFactory
            .buildTask(PasswordService.class, service -> service.getPasswordHistoricalNumber())
            .call()
            .done(result -> {
                String newValue = resetPasswordListOfValidations.getValue().replace("X", String.valueOf(result));
                resetPasswordListOfValidations.setValue(newValue);
            }).fail(result -> {});
    }
```

Índice

- Definición
- Java 8
  - Uso
  - Functional interface
  - Ejemplo en Enact
  - Performance
  - Best practices
- Links de interés

Performance compared to anonymous classes

When application is launched each class file must be loaded and verified.

Anonymous classes are processed by compiler as a new subtype for the given class or interface, so there will be generated a new class file for each.

Lambdas are different at bytecode generation, they are more efficient, used invokedynamic instruction that comes with JDK7.

For Lambdas this instruction is used to delay translate lambda expression in bytecode untill runtime. (instruction will be invoked for the first time only)

As result Lambda expression will becomes a static method(created at runtime). (There is a small difference with stateles and statefull cases, they are resolved via generated method arguments)

Best practices

- Single level of abstraction principle (link)
- Usar parametros tipados solo cuando agreguen claridad al cuerpo de la función.
- Abstract class or interface? Polémico. Siempre que se pueda usar una interfaz con default methods, hacerlo ya que es la opción menos restrictiva (posibilita herencia múltiple).

Links de interés

- Lambda expressions - Stack Overflow - (link)
- Commit con un ejemplo en Enact (link)
- Single level of abstraction principle (link)
- Streams - Stack Overflow - (link)

Changüí

Java 8 Functions and anonymous methods

Java 8 Function Interface

```
@FunctionalInterface
public interface Function<T,R>{
    default <V> Function<T,V> andThen(Function<? super R,? extends V> after){/*...*/}

    R apply(T t)

    default <V> Function<V,R> compose(Function<? super V,? extends T> before){/*...*/}

    static <T> Function<T,T> identity(){/*...*/}
}
```

Java 8 anonymous methods

Comparator con lambda expression

```
Collections.sort(
    personList,
    (p1, p2) -> p1.getFirstName().compareTo(p2.getFirstName())
);
```

Java 8 anonymous methods

Comparator con lambda expressions (II)

```
Collections.sort(
    personList,
    Comparator
      .comparing(p -> p.getFirstName)
      .thenComparing(p -> p.getLastName))
);
```

Comparator con métodos anónimos

```
Collections.sort(
    personList,
    Comparator
      .comparing(Person::getFirstName)
      .thenComparing(Person::getLastName))
);
```

Java 8 anonymous method

```
@FunctionalInterface
public interface Function<T,R>{
    default <V> Function<T,V> andThen(Function<? super R,? extends V> after){/*...*/}

    R apply(T t)

    default <V> Function<V,R> compose(Function<? super V,? extends T> before){/*...*/}

    static <T> Function<T,T> identity(){/*...*/}
}
```

Changüí II

Java 8 Streams

Java 8 Streams

```
Stream
    .of("apple", "banana", "pear", "kiwi", "orange")
    .filter(s -> s.contains("a"))
    .map(String::toUpperCase)
    .sorted()
    .forEach(System.out::println);
```

Output:

APPLE
BANANA
ORANGE
PEAR