TP11: JSON, réflexion et annotations

Exercice 1 - JSON Encoder

1.
```java
public static String toJSON(Record record) {
    return Arrays.stream(record.getClass().getRecordComponents())
            .map(component -> "\"" + component.getName() +
                    "\":" + escape(invoke(record, component.getAccessor())))
            .collect(Collectors.joining(", ", "{", "}"));
}

private static Object invoke(Record record, Method accessor) {
    try {
        return accessor.invoke(record);
    } catch (IllegalAccessException e) {
        throw new IllegalStateException("method not accessible", e);
    } catch (InvocationTargetException e) {
        var cause = e.getCause();
        switch (cause) {
            case RuntimeException exception -> throw exception;
            case Error error -> throw error;
            default -> throw new UndeclaredThrowableException(e);
        }
    }
}
```

2.
```java
@Target(ElementType.RECORD_COMPONENT)
@Retention(RetentionPolicy.RUNTIME)
public @interface JSONProperty {
    String value();
}
public static String toJSON(Record record) {
    return Arrays.stream(record.getClass().getRecordComponents())
            .map(component -> "\"" + name(component) + "\":" +
escape(invoke(record, component.getAccessor())))
            .collect(Collectors.joining(", ", "{", "}"));
}
private static String name(RecordComponent recordComponent) {
    var name = recordComponent.getName();
    var annotation = recordComponent.getAnnotation(JSONProperty.class);
    return (annotation == null || annotation.value().isEmpty()) ? name :
annotation.value();
}
private static String name(RecordComponent recordComponent) {
    var name = recordComponent.getName();
    var annotation = recordComponent.getAnnotation(JSONProperty.class);
    if (annotation == null) {
        return name;
    }
    var value = annotation.value();
    return value.isEmpty() ?
```

```
            name.replace('_', '-') :
            value;
}
```

3. Car il renvoie un tableau et qu'un tableau est mutable.

4.
   ```
   private static final ClassValue<RecordComponent[]> CACHE =
           new ClassValue<>() {
               protected RecordComponent[] computeValue(Class<?> type) {
                   return type.getRecordComponents();
               }
           };
   ```

5.
   ```
   private static final ClassValue<List<Function<Record, String>>> CACHE =
           new ClassValue<>() {
               protected List<Function<Record, String>> computeValue(Class<?>
   type) {
                   return Arrays.stream(type.getRecordComponents())
                           .<Function<Record, String>>map(component -> {
                               var key = "\"" + name(component) + "\":";
                               var accesor = component.getAccessor();
                               return record -> key + escape(invoke(record,
   accesor));
                           })
                           .toList();
               }
           };

   public static String toJSON(Record record) {
       return CACHE.get(record.getClass()).stream()
               .map(function -> function.apply(record))
               .collect(Collectors.joining(", ", "{", "}"));
   }
   ```

Steve Chen