Java 8 Guidelines

Java 8 Guidelines Overview

- Lambdas Guidelines
- Streams Guidelines
- Date and Time – Migration Tips
- Guava – Migration Tips

Lambda Guidelines

Rules of thumb – Lambdas

- **Lambdas should be short, ideally be a one-liner**

```
i -> i % 2 == 0
str -> str.isEmpty()
```

- Use **descriptive names** for paramaters when appropriate

```
p -> p.isAdult()
person -> person.isAdult()
```

- Use **explaining variables** to communicate the intention of lambdas:

```
Predicate<Person> isAdult = person -> person.getAge() >= 18;
Predicate<Integer> isEven = i -> i % 2 == 0;
```

- **Prefer method references** instead of lambda

```
person -> person.isAdult()    =>    Person::isAdult
```

Rules of thumb – Lambdas

- **Avoid state modifications in lambdas**

- **Avoid heavy calculations in lambdas**
  - Limit the code to 5-10 lines

- **Avoid Lambdas that throw checked Exceptions**
  - no exception throwing code

Why?

Functional Programming assumes functional behaviour =>
transformation from input(s) to outputs without side effects

Streams Guidelines

Rules of thumb – Streams

- **Import Collectors statically to improve readability**
- **Show «Pipeline» in layout of source code**
- **explaining variable may increase readability**

```
public static List<Stream> transform(List<Person> persons) {
    Predicate<Person> allowedToDrive = person -> person.isAdult() &&
                                                  person.hasDrivingLicense();
    return persons.stream()
                  .filter(allowedToDrive)
                  .map(String::toUpperCase)
                  .collect(toList());
}
```

Rules of thumb – Streams

- **Avoid concatenating of to many functions,** preferable limits:
  - `0-3 * filter()`
  - `0-2 * map()`
  - `1   * collect() / reduce()`

- **Avoid parallelStream()**
  - Only very large data sets may be processed faster
  - All parallel operations share the same thread pool *
  - JEE server: parallel is mapped to sequential
  - Potentially unexpected / unpredictable ordering (`forEach()` and `parallelSteam()`)
  - http://zeroturnaround.com/rebellabs/java-parallel-streams-are-bad-for-your-health/

Rules of thumb – Streams

- **Public method should not return Streams**
- **Streams can only be consumed once => Should be consumed in the same function as they are produced normally**

```
IntStream stream = IntStream.of(1, 2,3,4,5,6,7,8,9,10);
stream.forEach(System.out::println); // 1,2 ..., 10


stream.filter(i -> i % 2 == 0)
        .forEach(System.out::println); // Exception
```

- **if you need to access an index, avoid using a stream**

Best practices All In One

```
Predicate<FreeUnit> containsPhoneNumber =
                        unit -> unit.getNumbers().contains(phoneNumber);


List<FreeUnit> myUnlimitedUnits = getFreeUnits(user).stream()
                                      .filter(FreeUnit::isUnlimited)
                                      .filter(containsPhoneNumber)
                                      .collect(toList());


String numbers = unit.getNumbers().stream()
                                       .map(PhoneNumber::toString)
                                       .collect(joining(", "));
```

A) **Predicate explaining variable**,
   lambda with intention revealing variable name

B) **Pipeline-Layout**

C) static import ***Collectors. toList***()/***joining***()

D) **method reference**

Date and Time – Migration Tips

Joda -> JDK 8 Date And Time

1. comment out JodaTime dependency in build files
2. check all syntax errors
   - Correct import the like-named classes for JSR 310
     - org.joda.time.* ——> java.time.*
3. perform the following transformations

JODA-TIME	JDK 8 Date and Time API
new LocalDate()	LocalDate.now()
new LocalDate(year, month, day)	LocalDate.of(year, month, day)
LocalDate.fromDateFields(javautildate)	LocalDate.from(javautildate.toInstant())
localDate.getMonthOfYear()	localDate.getMonthValue()

Joda -> JDK 8 Date And Time

Transformations continued:

JODA-TIME	JDK 8 Date and Time API
DateTimeFormatter.parseLocalDate("formatted_date")	LocalDate.from(DateTimeFormatter.parse("formatted _date"))
localdate.toString(dateTimeFormatter)	localDate.format(dateTimeFormatter)
DateTimeFormat.forPattern("pattern")	DateTimeFormatter.ofPattern("pattern")
DateTimeFormatter.print(localdate)	DateTimeFormatter.format(localdate)
"MMMMM" *// 5 M = formatting pattern for full month name*	"MMMM" *// 4 M = formatting pattern for full month name*

http://blog.joda.org/2014/11/converting-from-joda-time-to-javatime.html

Guava – Migration Tips

Guava -> JDK 8

1. Guava covers many of the same functionalities as Java 8
2. "If something is in the JDK, we will use it in the JDK"

Guava	JDK 8
Iterables	Stream
Predicate	Predicate
Optional	Optional

Joiner	Collectors.joining(", ")
	String.join(", ", elements)

Predicates Migration

- Guava Predicates use static methods for combination of several Predicates

```
Predicates.and(notNull, hasId)
```

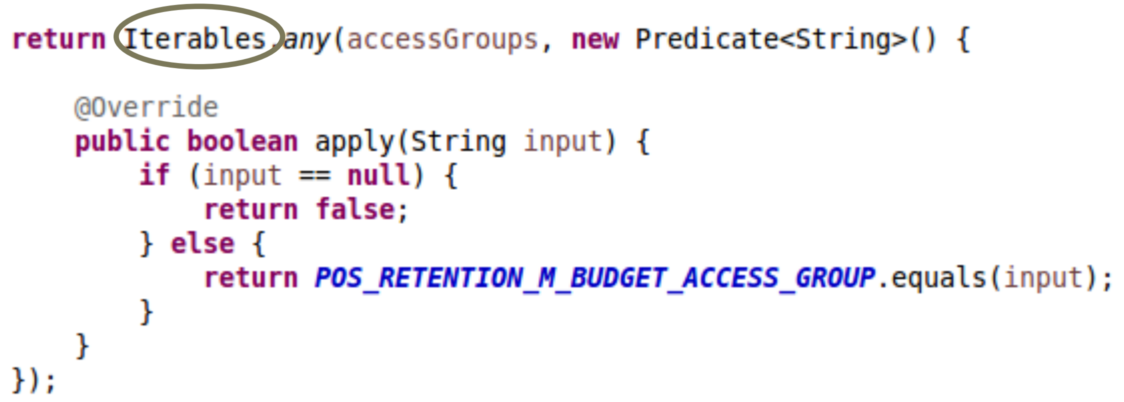
- Java Predicates use default methods on the Predicate Interface

```
notNull.and(hasId)
```

- Guava also defines lots of small Predicates (notNull, alwaysTrue etc). Java does not define these. Explained in Detail on:

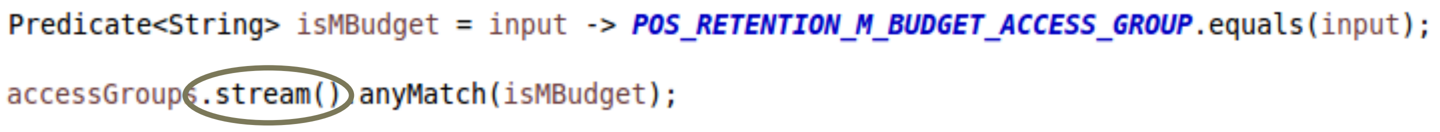
  http://stackoverflow.com/questions/26549659/built-in-java-8-predicate-that-always-returns-true

Iterables Migration



```
return Iterables.any(accessGroups, new Predicate<String>() {

    @Override
    public boolean apply(String input) {
        if (input == null) {
            return false;
        } else {
            return POS_RETENTION_M_BUDGET_ACCESS_GROUP.equals(input);
        }
    }
});
```



```
Predicate<String> isMBudget = input -> POS_RETENTION_M_BUDGET_ACCESS_GROUP.equals(input);

accessGroups.stream().anyMatch(isMBudget);
```