AARHUS UNIVERSITET

Software Engineering and Architecture

Design Patterns
and
Lambda in Java8

AARHUS UNIVERSITET

My PhD Supervisor said…

- *Design patterns are just deficiencies in the underlying programming language.* [Ole Lehrman Madsen]

- There is some truth in that statement…

- Let us have a look at Java8 and what it can do…

AARHUS UNIVERSITET

Lambda in Java 8

- @FunctionalInterface
  - Annotation on an interface signalling that the interface *has only one method* and can thus be treated as a *function*

```
@FunctionalInterface
interface RateStrategy{
  int calcTime(int value);
}
```

- *Any interface with only single abstract method is a functional interface*
  - Read: The java compiler adds the annotation automatically…

AARHUS UNIVERSITET

Strategy Pattern

- Many Strategies express *an algorithm in only a single method* and can thus be expressed faster using lambda's.

```
class PayStation {
  PayStation(RateStrategy rs) {
    this.rs = rs;
  }
  private RateStrategy rs;
  void addPayment(int coin)
    this.coin = coin;
  }
  private int coin;
  int readDisplay() {
    return rs.calcTime(coin)
  }
}
```

```
public static void main(String[] args) {
  System.out.println("=== Patterns in a Java8 Context ===");
  PayStation ps;
  // AlphaTown
  ps = new PayStation( v -> v/5*2 );
  ps.addPayment(5);
  System.out.println(" AlphaTown: 5 cent = " + ps.readDisplay() + " min");
  // TestTown
  ps = new PayStation( v -> v );
  ps.addPayment(5);
  System.out.println(" TestTown: 5 cent = " + ps.readDisplay() + " min");
```

AARHUS UNIVERSITET

java.util.function

Function<T,R>	Represents a function that accepts one argument and produces a result.

```
// Paystation but with a Function instead
class PayStationF {
  PayStationF(Function<Integer, Integer> rs) {
    this.rs = rs;
  }
  private Function<Integer, Integer> rs;
  void addPayment(int coin) {
    this.coin = coin;
  }
  private int coin;
  int readDisplay() {
    return rs.apply(coin);
  }
}
```

Our RateStrategy is just a
f: x -> y
Refactor to use
java.util.function.Function

```
// TestTown
psF = new PayStationF( v -> v );
```

What about ‘large’ strategies?

- BetaTown RateStrategy – many more lines of code…
- You can use *method references*
  - BetaRateStrategy::apply

```
class BetaRateStrategy {
  public static Integer apply(Integer value) {
    // VERY LONG ALGORITHM HERE

    // ....

    // ....

    return value + 1000;
  }
}
```

```
// BetaTown
psF = new PayStationF( BetaRateStrategy::apply );
psF.addPayment(5);
System.out.println("func BetaTown: 5 cent = " + psF.readDisplay()
```

Quite a few ways of referring to functions, this is just one way…

AARHUS UNIVERSITET

Strategy

- A strategy that just operates GameImpl methods

Consumer<T> Represents an operation that accepts a single input argument and returns no result.

- Consumer has single method
  - void accept(T t);
- Ala
  - WorldLayoutStrategy implements Consumer<GameImpl>
    - Accept method would then call back to gameimpl and set up world.

- So – should we remove all strategies and replace with these weird fellows???

AARHUS UNIVERSITET

Morale: Patterns are Communication

- Maybe this is just mental enertia on my behalf, but…
- *Expressing the interfaces are explicit documentation…*
- I think you may loose some of that if your design just becomes a bunch of Function, Consumer, BiFunction, etc.
- But – perhaps just a matter of training…

AARHUS UNIVERSITET

Command Pattern

AARHUS UNIVERSITET

Command Pattern

- Command is the Pattern to *convert an method invocation into an object.*
  - But hey – that is the very definition of a lambda!
    - Function pointer – a reference to a specific method, that is a first class object to be handled explicitly by the language…

- However, only really works for ‘execute()’ and not for ‘undo()’
  - Two methods in the Command interface means that is not a functional interface and cannot be considered a lambda.

AARHUS UNIVERSITET

Iterator

The objectification of iteration

AARHUS UNIVERSITET

Java Iterator

- Ask collection for its *iterator*, use that to visit every elem.

```
// Demonstration iterator
public void sweepMapIterator() {
  System.out.println("-- Sweeping a map / Iterator --");
  Map<Position, City> map = new HashMap<>();
  map.put(new Position(1,1), new City()); // The red city
  map.put(new Position(4,1), new City()); // The blue city

  // Sweeping a map, by iterator
  Iterator<Position> i = map.keySet().iterator();
  while( i.hasNext()) {
    Position p = i.next();
    City element = map.get(p);
    System.out.println("Processing "+element);
  }
}
```

AARHUS UNIVERSITET

For each loop

- This was cumbersome, so syntactic sugar was applied.

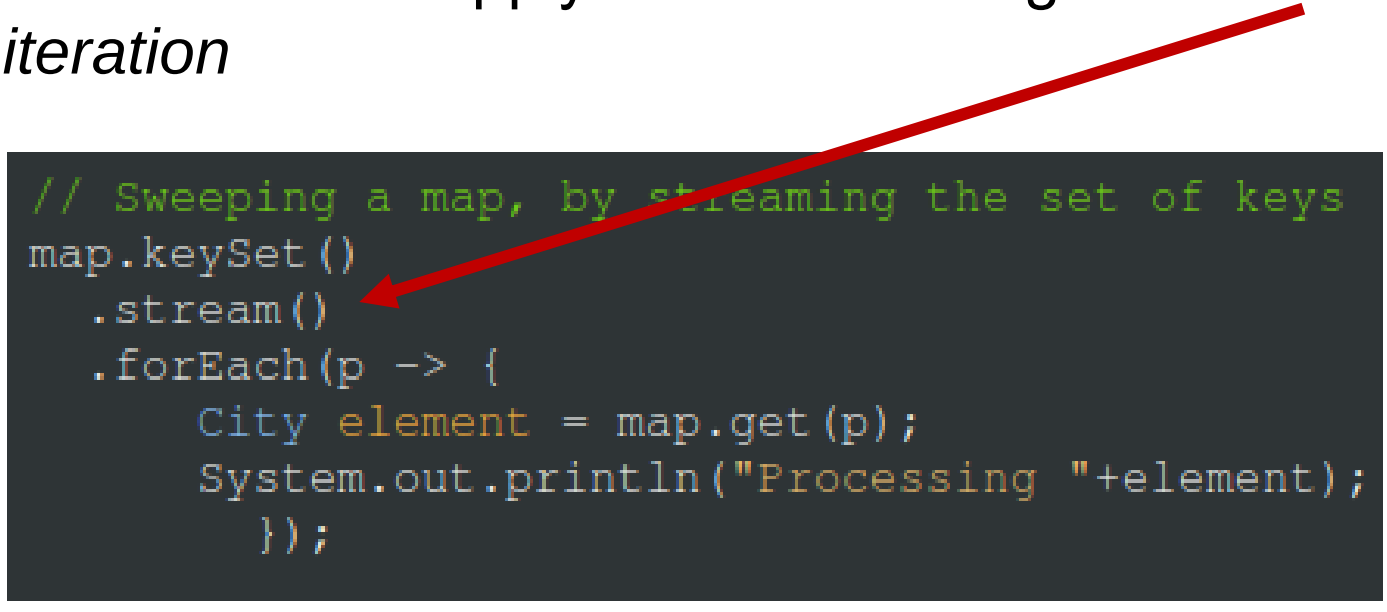
```
// Sweeping a map, by iteration on the set of keys
for (Position p: map.keySet()) {
  City element = map.get(p);
  System.out.println("Processing "+element);
}
```

- Both way, however is *external iterators*
  - The iterator ‘surrounds’ the collections

AARHUS UNIVERSITET

Streaming libraries

- Lambda functions allow to just supply the collection with the ‘function to apply’ and then change to *internal iteration*



```
// Sweeping a map, by streaming the set of keys
map.keySet()
   .stream()
   .forEach(p -> {
        City element = map.get(p);
        System.out.println("Processing "+element);
          });
```