Alain Lompo
Senacor Technologies AG

Designing function families and bundles with Java's Behaviours parameterization and lambdas

About me

- I am a software developer at Senacor
- Building e-banking solutions using
  - Java (JEE/Spring boot) in the backend
  - Angular/React on the frontend
- Available at: @alainlompo

Plan

- Motivation
- Functional programming to the rescue
- Behaviors parameterization
- Designing families of functions
- Applications and Demos
- Wrap - up

MOTIVATION

User requirements are sinking sand

- They always seem good at first
- But they always change later
- It happens generally during the course of implementation
- Sometimes even later

Blaming it on the methodology

- It does not matter which methodology is used, the problem will still be there
- We simply find out about it earlier or later depending on the methodology
- With agile methods we find about it earlier

Costs optimisation

- Software maintenance costs generally more than its initial implementation
- Behavior’s parameterization can help reduce these costs significantly

Keeping the developers happy

- Developers generally feel happy when
  - The task has been successfully implemented, tested and set as Ready for PROD (green check mark)
  - The task was not specified clearly enough so it is set in a « REQUIRES CLARIFICATIONS » or « BLOCKED » status
  - In both case they can move forward with new (and exciting) tasks and live happily ever after.

Functionnal programming to the rescue

Main benefits of functional programming

- Finish on time and meet deadlines
  - Reduces time to market for your projects
- Write correct code
  - Avoid mutability and state handling issues
  - Avoid null values handlind and NPEs
  - Avoid external iterations
  - Express intent and « the what » rather than « the how »

Main benefits of functional programming

- Handle complexity
  - …with simpler code
  - Leads to using more advanced algorithms and providing better functionalities
- Efficient and scalable code
  - Easier to parallelize code
  - Better abstractions for writing reactive code
  - Better abstractions for writing asynchronous code

Objects vs functions

- In OOP everything is an object
- Sometime we simply need to use a functionality without needing a whole class
- With functional programming, functions become first class citizens

Passing behaviors through interfaces

```
RefinedFinancialService refinedFinancialService = new RefinedFinancialService();
double newAmount = refinedFinancialService.computeNewAmount(amount, interestRate, bonus,
        new AmountComputable() {
    @Override
    public double computeNewAmount(double amount, double interestRate, double bonus) {
        return amount * (1.5 + interestRate) + bonus;
    }
});
```

Core concepts

- Lambdas expressions
- Functional interfaces

Lambdas expressions

- With the OOP approach we could pass behaviors through interfaces

- But to use them we had to:
  - Create the interface
  - Create a class that implements the interface
  - Define a method with a signature that takes the interface as parameter
  - Call the method an give it an object that is an instance of a class that implements the interface

Lambdas expressions

- What if we could find a way to avoid all that extra stuff and just pass the action we want done?

Functions as values

- How can we define functions as values?
- So that we could associate them (the functions, not the result of their execution) to variables and reuse them?
- Lambdas expressions allow us to do that

Lambdas expressions

```
lambda1 = () -> System.out.println(« Hello world »);

Lambda2 = (amount, interestRate) -> amount * (1 + interestRate);

Lambda3 = n -> 1 + 1/n;
```

- Let's see how to build them

Functional interfaces

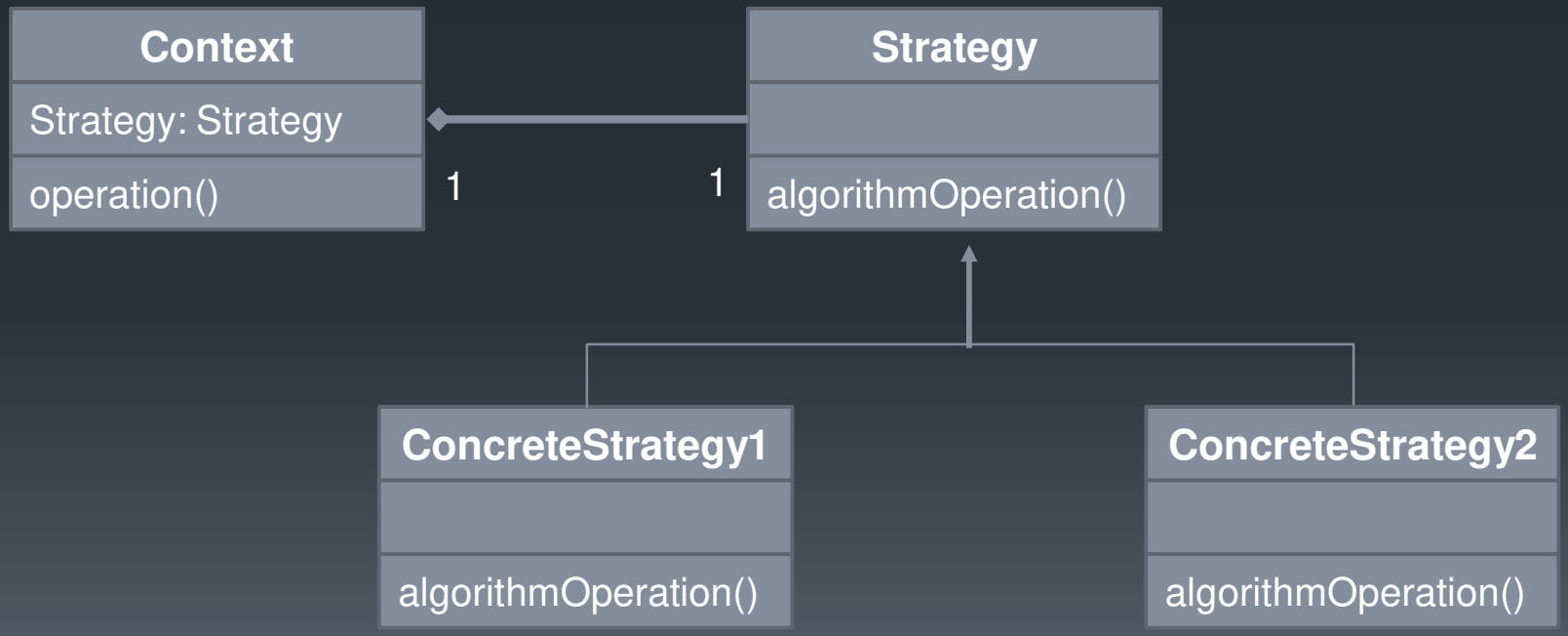
```
public interface Runnable {
   void run();
}
```

```
@FunctionalInterface
public interface Updatable {
   public void update(Resource
   resource);
}
```

Behaviors parameterization (Demos)

Designing families and bundles of functions

Using the strategy pattern



Parameterizing a family of behaviors

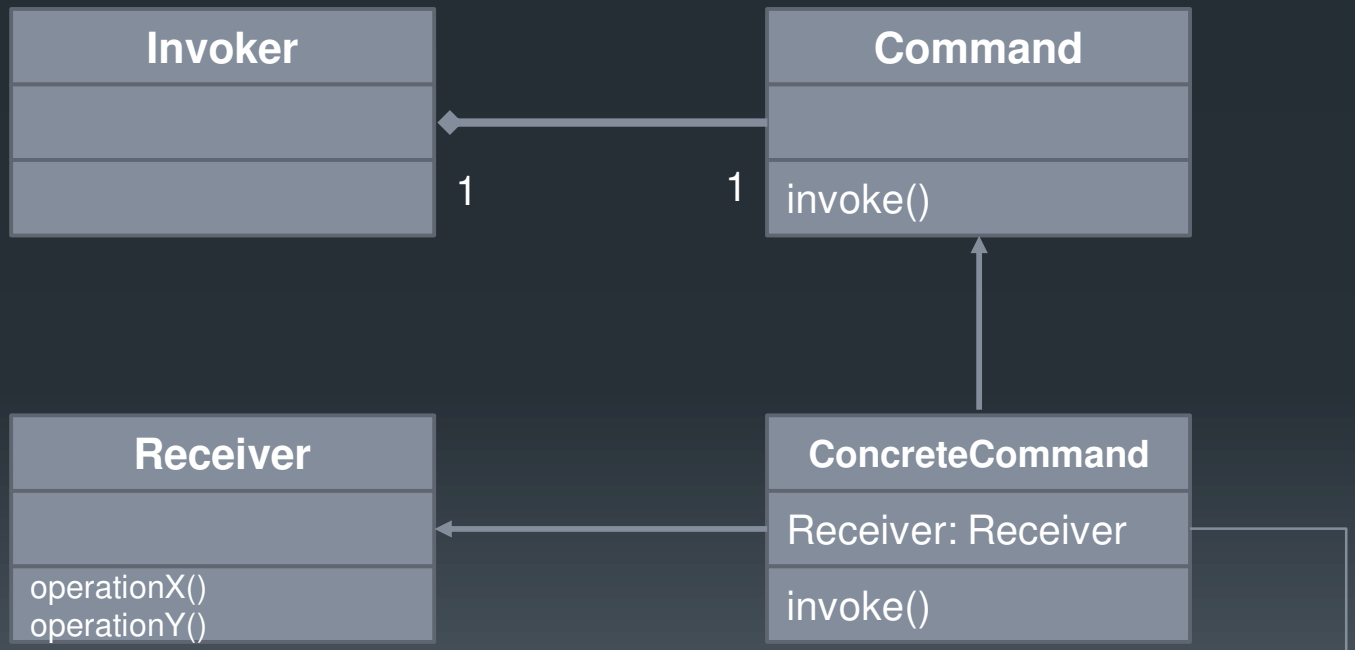
- The strategy pattern is useful if the application needs to choose between several algorithms or parts of algorithms

- Example:
  - Several tasks are similar except they differ in a small subtask
  - With strategy with define the common part and parameterize the varying subtask
  - Applying it with functional programming we can parameterize a familiy of functions

The strategy interface as functional interface

- The strategy interface has only one method
- It is therefore a good candidate to be used as a functional interface

using the command design pattern




```
void invoke {
    receiver.operationX();
}
```

Parameterizing a family of behaviors

- The command pattern
  - Describes a way to represent actions in an application
  - Used to store « unit of processing » that can be later re-invoked
    - For example to make a revert
  - Collections of command are often used to specify steps of operations that the user can choose from

The command operation as functional interface

- The command operation interface has only one method
- It is therefore a good candidate to be used as a functional interface

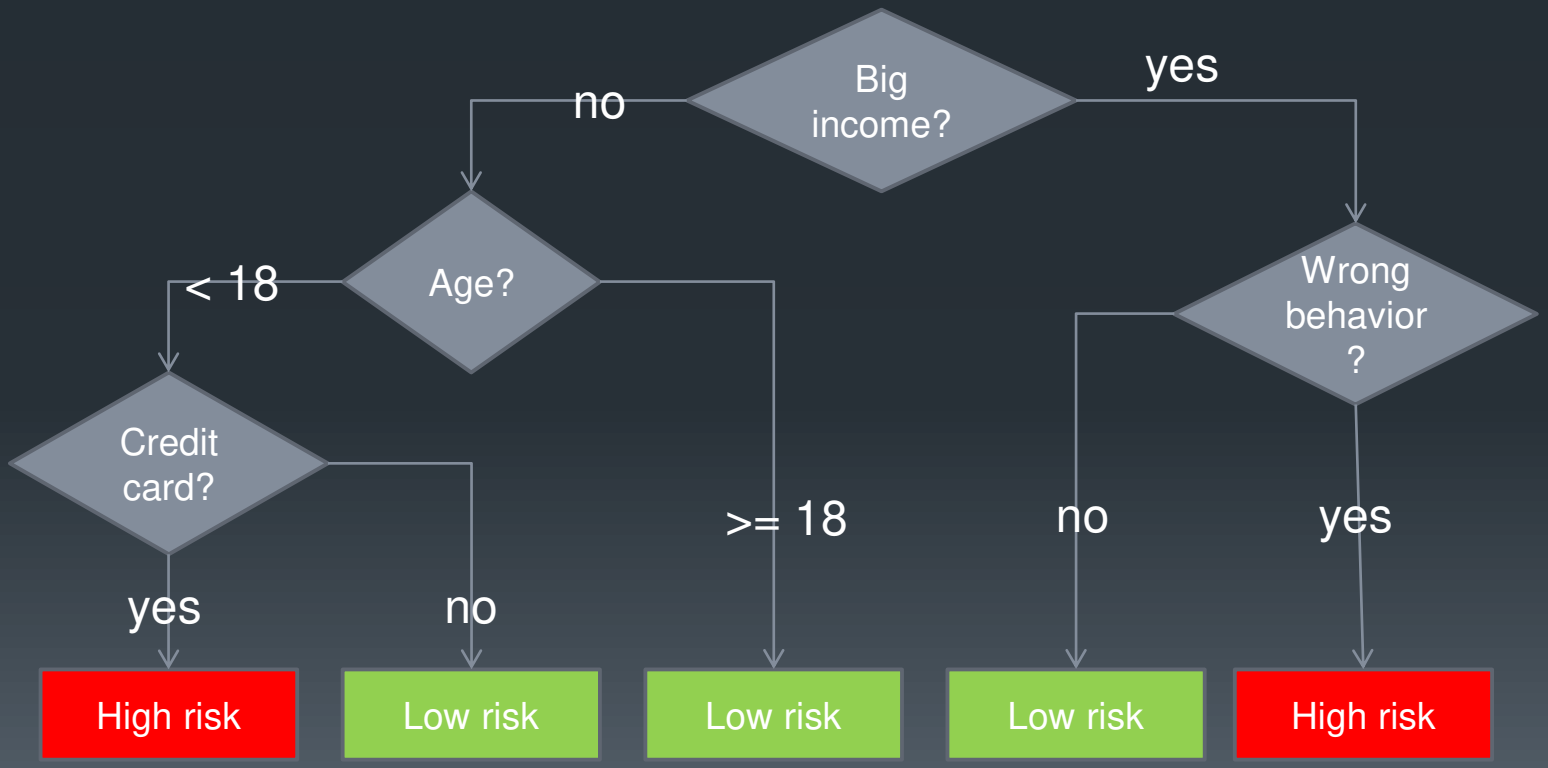
Receiver component and mutable state

- In OOP the receiver is mutable
- It holds somewhat the state and updates it after each command (when necessary)
- In functional programming with DO NOT mutate state
- But we can manage the return value of each command.

Decision trees

- Very popular in machine learning
- Can be used for:
  - Making decisions based on some datas
  - Classifying input into various categories
- The algorithm works with a tree that specifies
  - What properties of the data should be tested
  - What to be done with each possible answer
- The reaction can be another test or the final answer

Decision trees



Wrap-Up

Wrapping up

- User requirements are sinking sand
- OOP linked behavior to datas and made objects
- But sometimes the ceremony/rituals of OOP are meaningless
- Therefore FP comes to the rescue

Wrapping up

- Functional programming permits costs optimisations
- And helps keep developers happy
- It does that by enabling us to parameterize behaviors
  - By treating functions as values and allowing us to pass them as parameters
  - In java an interface with one method is treated as functional interface

Wrapping up

- We can therefore apply functional programming to behavioral design pattern to build paramaterized families and bundles of functions
- And doing so greatly reduce the necessity of modifying code with each fluctuation of user requirements

Questions?

Thank you !

Resources

- https://stackabuse.com/behavioral-design-patterns-in-java/