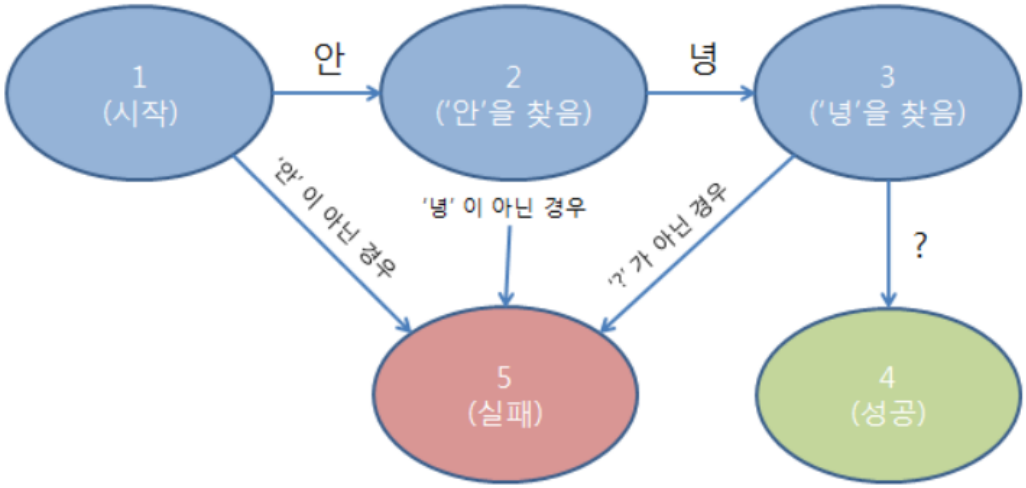
04.FSM Actor

(Finite State Machine)

, .

(finite-state machine, FSM) (finite automaton, FA; : finite automata) . . , . , (Current State) . (Event) , (Transition) . , .



- .
- .
- .
- .
- ().

FSM

.

OOP FSM

State Degisn Pattern .

AKKA FSM (https://doc.akka.io/docs/akka/2.5/fsm.html)

become()/unbecome() FSM .

HotSwapActor

,

.

- .
- foo .
- bar .
- .
- .

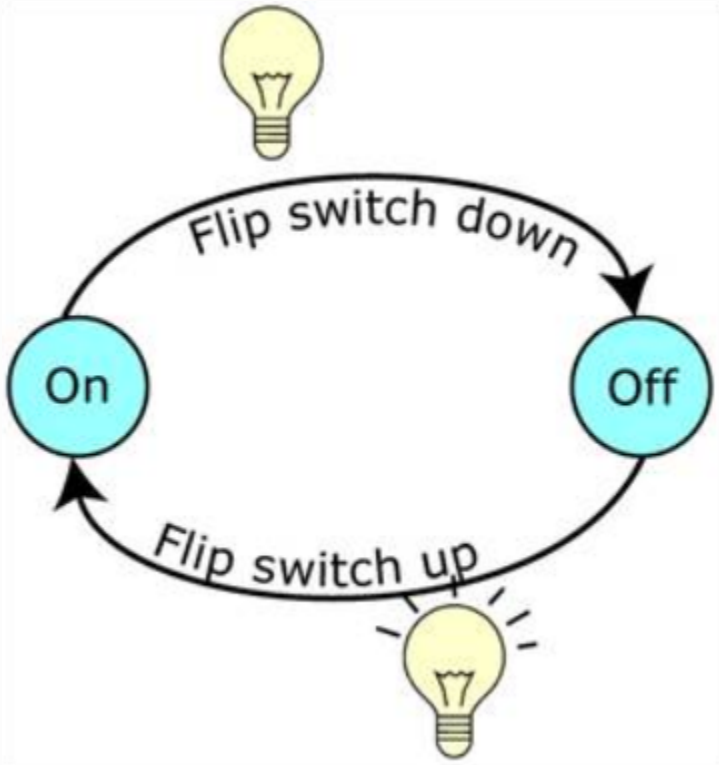
```
public class HotSwapActor extends AbstractActor {
  private AbstractActor.Receive angry;
  private AbstractActor.Receive happy;

  public HotSwapActor() {
    angry =
      receiveBuilder()
        .matchEquals("foo", s -> {
          getSender().tell("I am already angry?", getSelf());
        })
        .matchEquals("bar", s -> {
          getContext().become(happy);
        })
        .build();

    happy = receiveBuilder()
      .matchEquals("bar", s -> {
        getSender().tell("I am already happy :-)", getSelf());
      })
      .matchEquals("foo", s -> {
        getContext().become(angry);
      })
      .build();
  }

  @Override
  public Receive createReceive() {
    return receiveBuilder()
      .matchEquals("foo", s ->
        getContext().become(angry)
      )
      .matchEquals("bar", s ->
        getContext().become(happy)
      )
      .build();
  }
}
```

> *Actor API become()*
>
> *HowSwapActor ,*
>
> *.*

SwapperActor

Swapper

- ON / OFF .
- .
- (/)
- .
  - ON : OFF
  - OFF : ON



.

```
public class Swapper extends AbstractLoggingActor {
  @Override
  public Receive createReceive() {
    return receiveBuilder()
      .matchEquals(Swap, s -> {
        log().info("Hi");
        getContext().become(receiveBuilder().
          matchEquals(Swap, x -> {
            log().info("Ho");
            getContext().unbecome(); // resets the latest 'become' (just for fun)
          }).build(), false); // push on top instead of replace
      }).build();
  }
}

public class SwapperApp {
  public static void main(String[] args) {
    ActorSystem system = ActorSystem.create("SwapperSystem");
    ActorRef swapper = system.actorOf(Props.create(Swapper.class), "swapper");
    swapper.tell(Swap, ActorRef.noSender()); // logs Hi
    swapper.tell(Swap, ActorRef.noSender()); // logs Ho
    swapper.tell(Swap, ActorRef.noSender()); // logs Hi
    swapper.tell(Swap, ActorRef.noSender()); // logs Ho
    swapper.tell(Swap, ActorRef.noSender()); // logs Hi
    swapper.tell(Swap, ActorRef.noSender()); // logs Ho
    system.terminate();
  }
}
```

> *AKKA FSM*
>
> *Part FSM .*
>
> *:*
>
> - *https://doc.akka.io/docs/akka/current/fsm.html*
> - *https://doc.akka.io/docs/akka/current/persistence.html#persistent-fsm*