For the State exercise I decided to create a State Object called PersonState. It is subclassed by the Child, Adult or Pensioner. The PersonState will hold the state reference and will decide when to make the transition to another state based on the age.

My context Object is Person which on initialization setting the age to zero and current state to Child.

It has the method do_check which is delegated to the current_state reference. Also the incr_age method which decides when the state to be changed and sets the state to the Person.

```
require_relative 'person_state.rb'
class Person

  def initialize
    @age = 0
    @current_state = Child.new
  end

  def set_state(person)
    @current_state = person
  end

  def do_checks
    puts "Current state reference is: #{@current_state}"
    @current_state.do_checks
  end

  def incr_age
    @age+=1
    if (@age==18)
      set_state(Adult.new)
    end
    if (@age==65)
      set_state(Pensioner.new)
    end
  end

end
```

I also used the Template Method Pattern in the ***PersonState*** Abstract class the do_check method provides the algorithm for the check implemented by all of the derived classes. Basically my PersonState becomes a wrapper that delegates to its current_state reference.

In my main class now I have much simple code which loop through the and call do_check after each age increment.

Better solution is to make the transition of the state in the derived classes. So the main stays the same the difference in my Person is the age is accessible and two methods are changed . Where now in the do_check method I am passing the reference of the Person.

```
class Person
  attr_reader :age

  def initialize
    @age = 0
    @current_state = Child.new
  end

  def set_state(person)
    @current_state = person
  end

  def do_checks
    @current_state.do_checks(self)
  end

  def incr_age
    @age+=1
  end

end
```

This way I cannot use the Template Method Pattern. In do_check in the derived classes I am changing the state and performing the checks. So the Person State becomes complete wrapper.

```
require_relative 'person.rb'

#Person State is a wrapper
class PersonState

  def do_checks(person)
  end

end
```

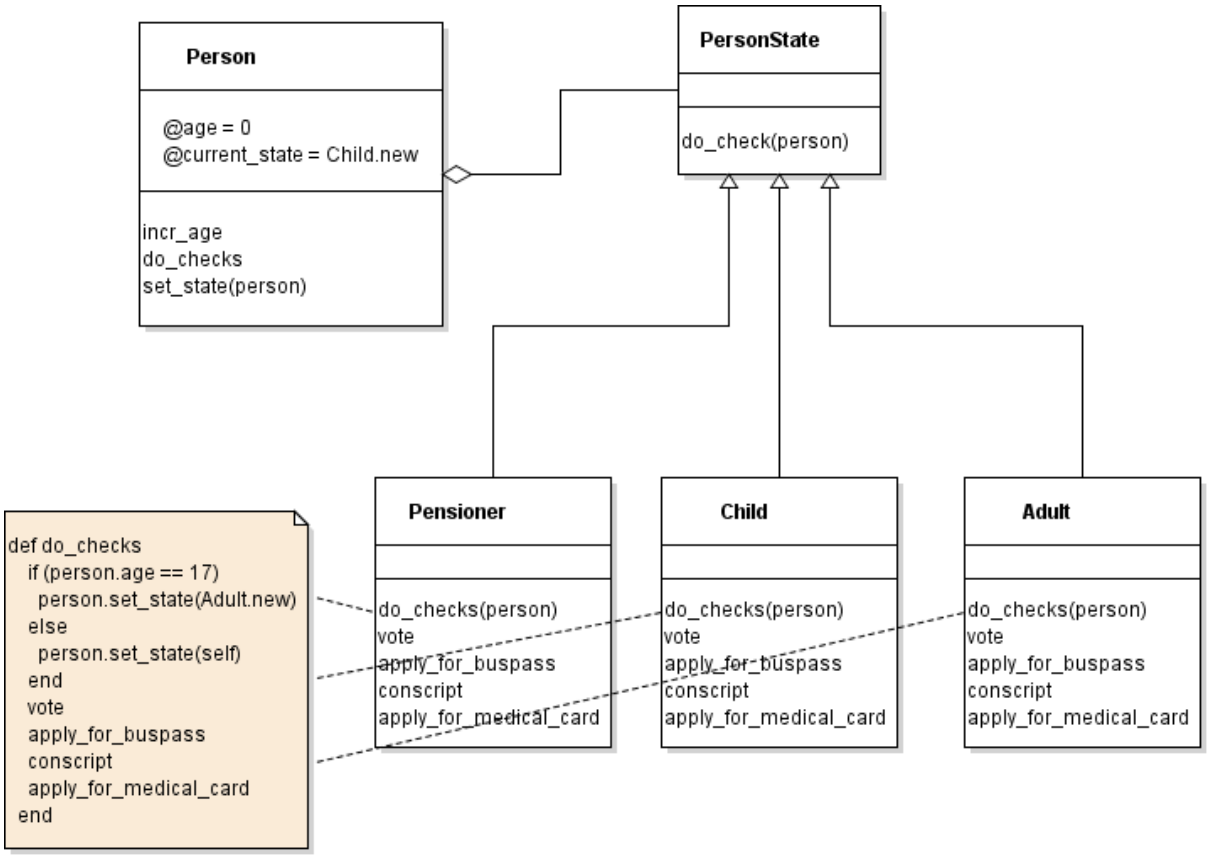
This is the Child implementation:

```
require_relative 'person_state.rb'
require_relative 'adult.rb'

class Child < PersonState

  def do_checks(person)
    if (person.age == 18)
      person.set_state(Adult.new)
    else
      person.set_state(self)
    end
```

```
    vote
    apply_for_buspass
    conscript
    apply_for_medical_card
  end

  def vote
    puts "Too young to vote"
  end

  def apply_for_buspass
    puts "Too young for a bus pass"
  end

  def conscript
    puts "Too young to be conscripted"
  end

  def apply_for_medical_card
    puts "Medical card granted"
  end

end
```

Similar for the rest Adult and Pensioner

This is the diagram for solution 2:



This is the class diagram for my solution 1: