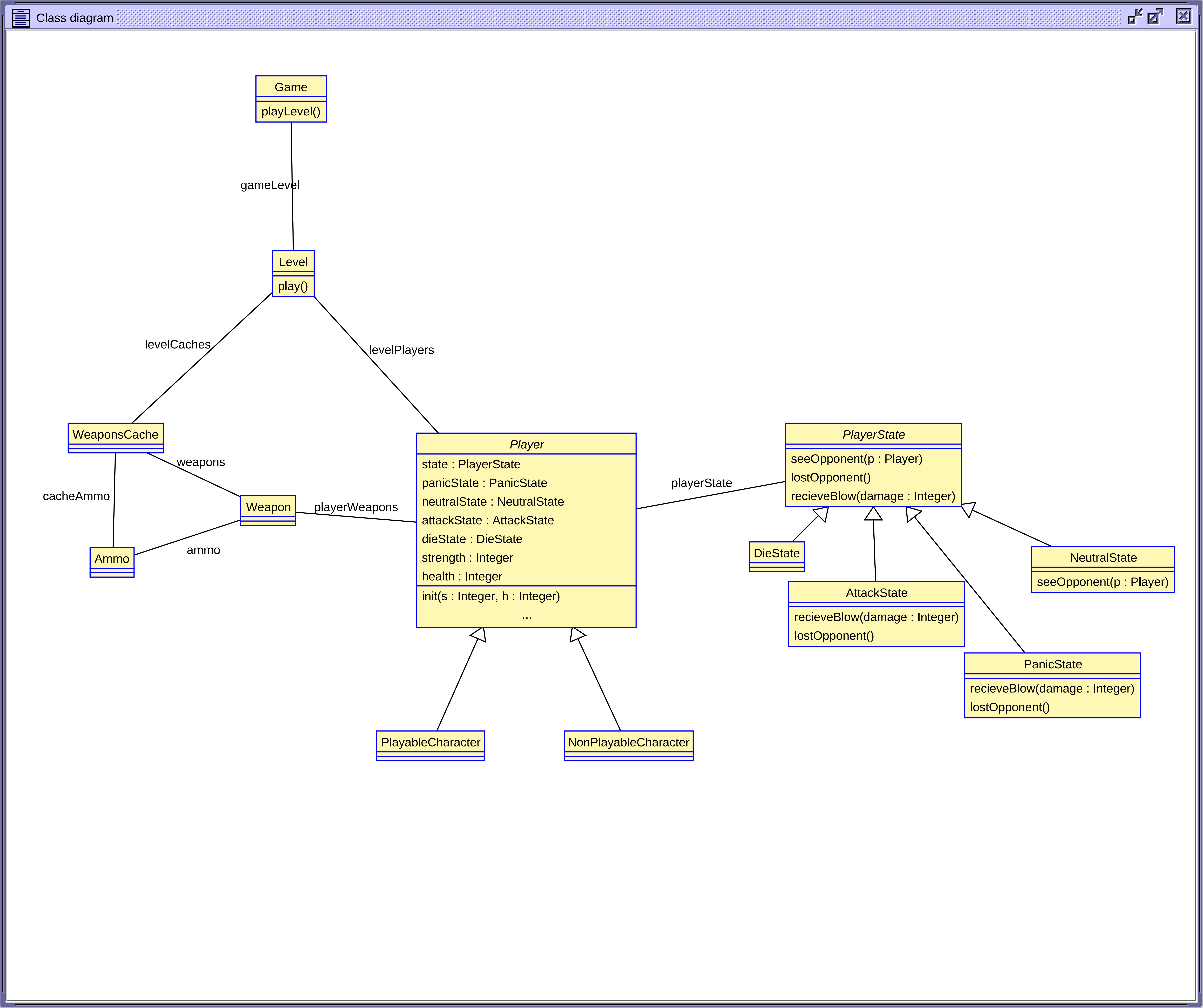
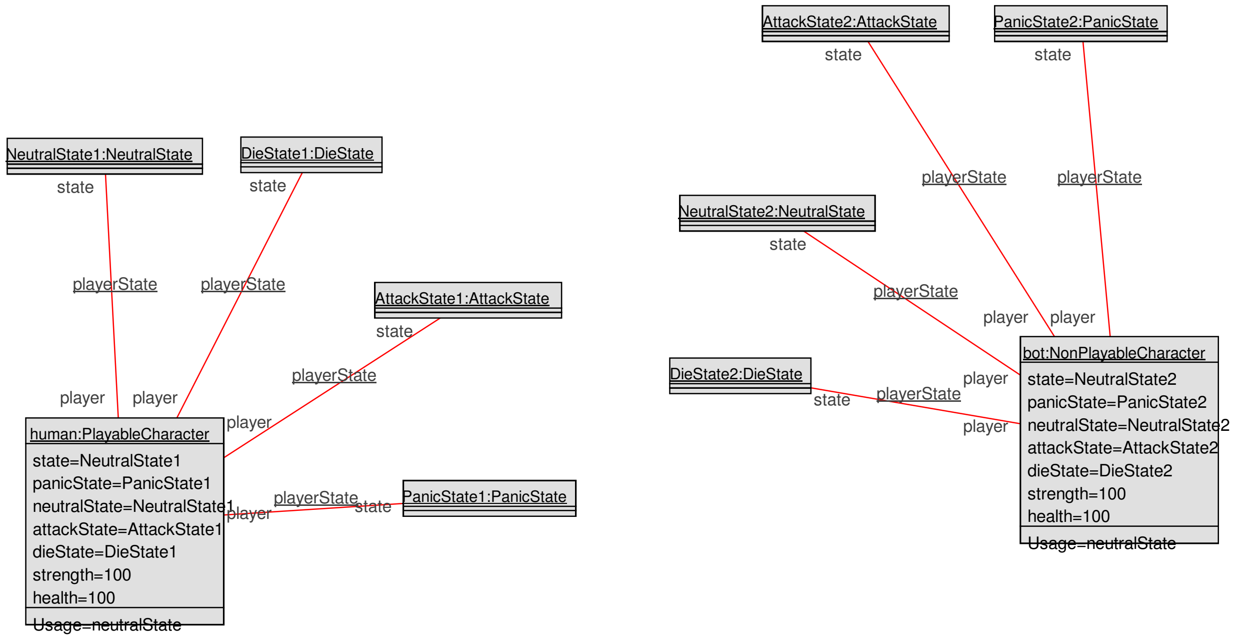
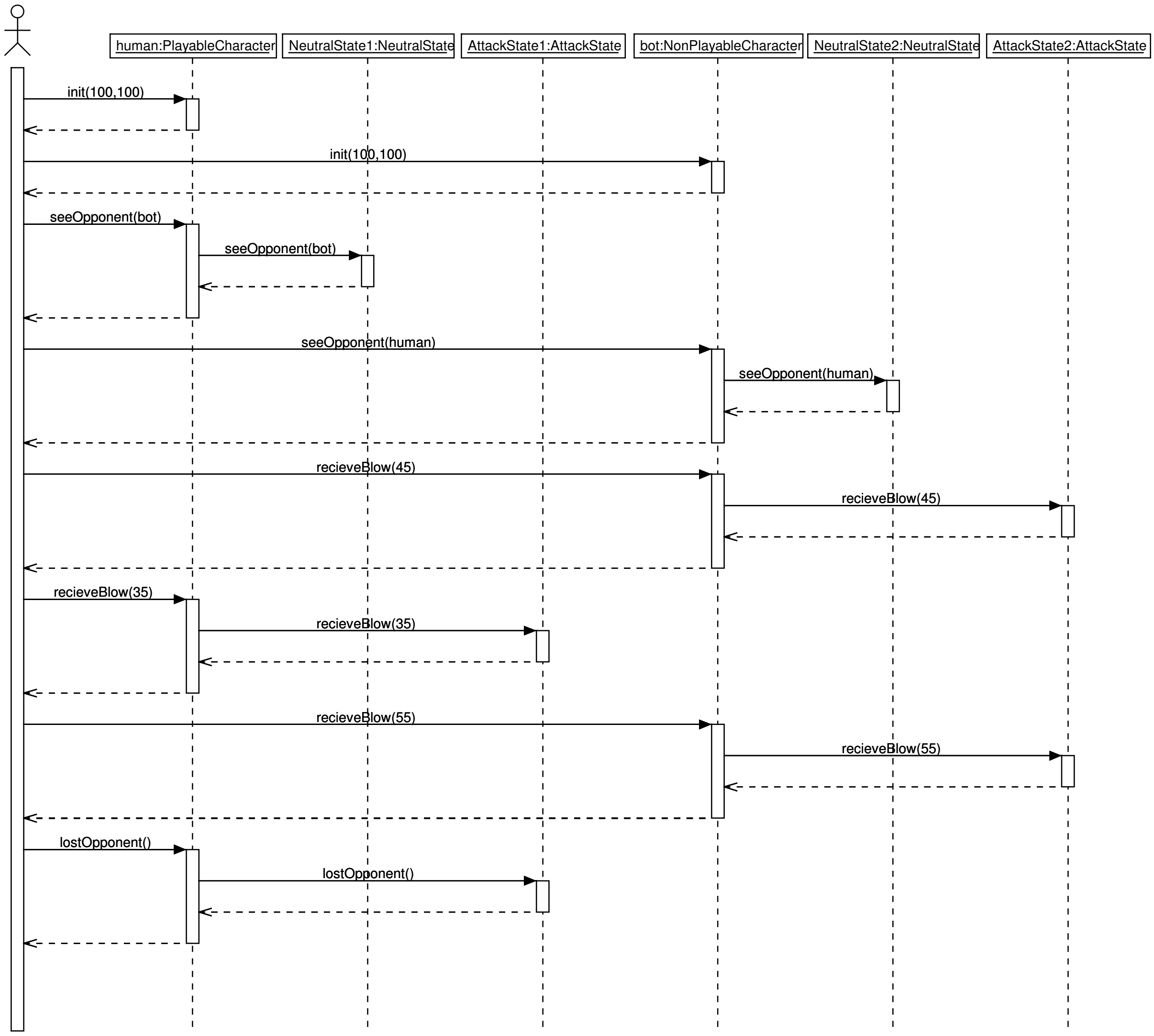
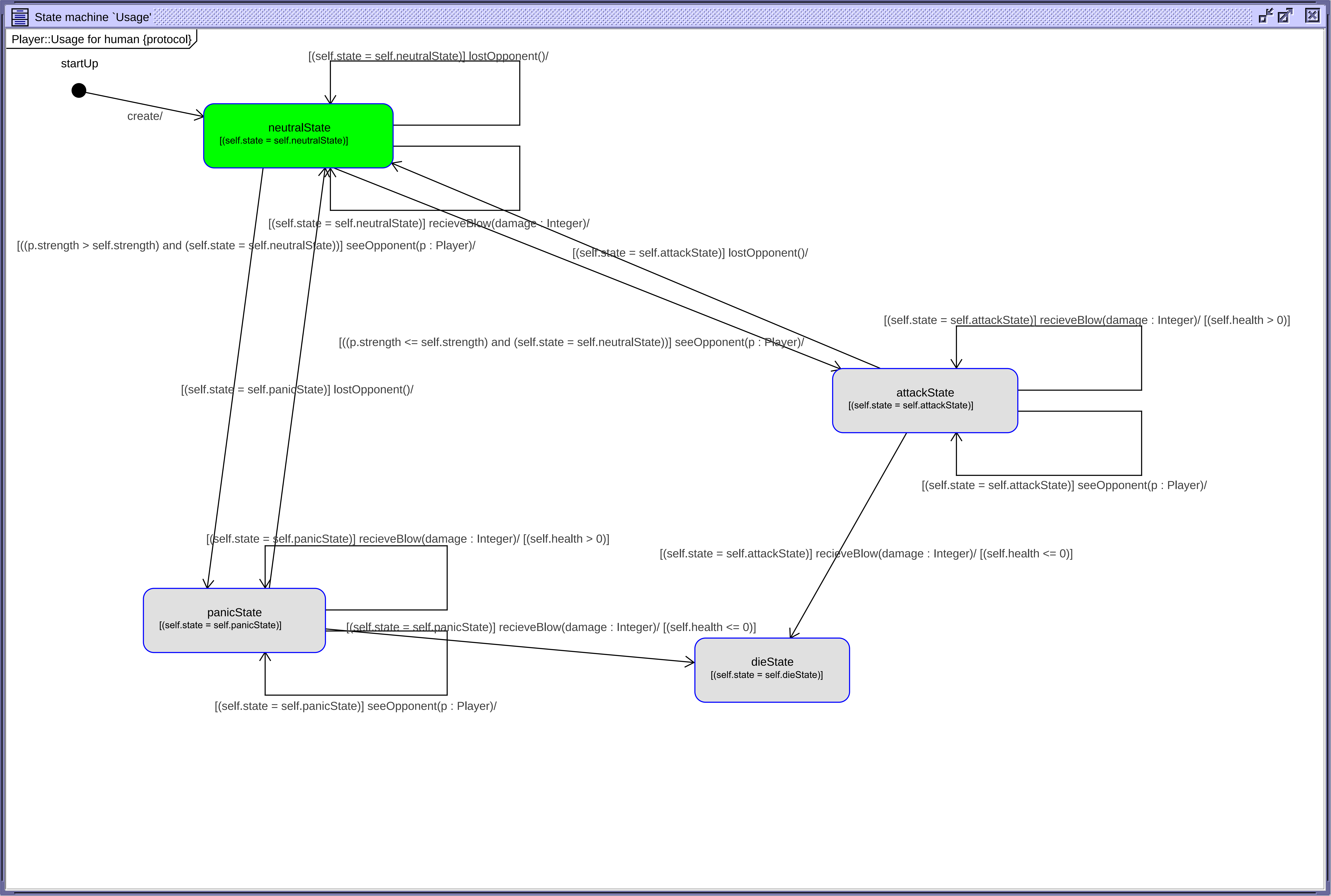

Class diagram
Game
playLevel()
gameLevel
Level
play()
levelCaches
levelPlayers
WeaponsCache
weapons
cacheAmmo
Weapon
playerWeapons
Ammo
ammo
Player
state : PlayerState
panicState : PanicState
neutralState : NeutralState
attackState : AttackState
dieState : DieState
strength : Integer
health : Integer
init(s : Integer, h : Integer)
...
playerState
PlayerState
seeOpponent(p : Player)
lostOpponent()
recieveBlow(damage : Integer)
DieState
NeutralState
seeOpponent(p : Player)
AttackState
recieveBlow(damage : Integer)
lostOpponent()
PanicState
recieveBlow(damage : Integer)
lostOpponent()
PlayableCharacter
NonPlayableCharacter


NeutralState1:NeutralState
DieState1:DieState
AttackState1:AttackState
PanicState1:PanicState
human:PlayableCharacter
state=NeutralState1
panicState=PanicState1
neutralState=NeutralState1
attackState=AttackState1
dieState=DieState1
strength=100
health=100
Usage=neutralState
state
playerState
player
AttackState2:AttackState
PanicState2:PanicState
NeutralState2:NeutralState
DieState2:DieState
bot:NonPlayableCharacter
state=NeutralState2
panicState=PanicState2
neutralState=NeutralState2
attackState=AttackState2
dieState=DieState2
strength=100
health=100
Usage=neutralState


human:PlayableCharacter
NeutralState1:NeutralState
AttackState1:AttackState
bot:NonPlayableCharacter
NeutralState2:NeutralState
AttackState2:AttackState
init(100,100)
init(100,100)
seeOpponent(bot)
seeOpponent(bot)
seeOpponent(human)
seeOpponent(human)
recieveBlow(45)
recieveBlow(45)
recieveBlow(35)
recieveBlow(35)
recieveBlow(55)
recieveBlow(55)
lostOpponent()
lostOpponent()


State machine `Usage'
Player::Usage for human {protocol}
startUp
create/
neutralState
[(self.state = self.neutralState)]
[(self.state = self.neutralState)] lostOpponent()/
[(self.state = self.neutralState)] recieveBlow(damage : Integer)/
[((p.strength > self.strength) and (self.state = self.neutralState))] seeOpponent(p : Player)/
[(self.state = self.attackState)] lostOpponent()/
[(self.state = self.attackState)] recieveBlow(damage : Integer)/ [(self.health > 0)]
[((p.strength <= self.strength) and (self.state = self.neutralState))] seeOpponent(p : Player)/
attackState
[(self.state = self.attackState)]
[(self.state = self.panicState)] lostOpponent()/
[(self.state = self.attackState)] seeOpponent(p : Player)/
[(self.state = self.panicState)] recieveBlow(damage : Integer)/ [(self.health > 0)]
[(self.state = self.attackState)] recieveBlow(damage : Integer)/ [(self.health <= 0)]
panicState
[(self.state = self.panicState)]
[(self.state = self.panicState)] recieveBlow(damage : Integer)/ [(self.health <= 0)]
dieState
[(self.state = self.dieState)]
[(self.state = self.panicState)] seeOpponent(p : Player)/

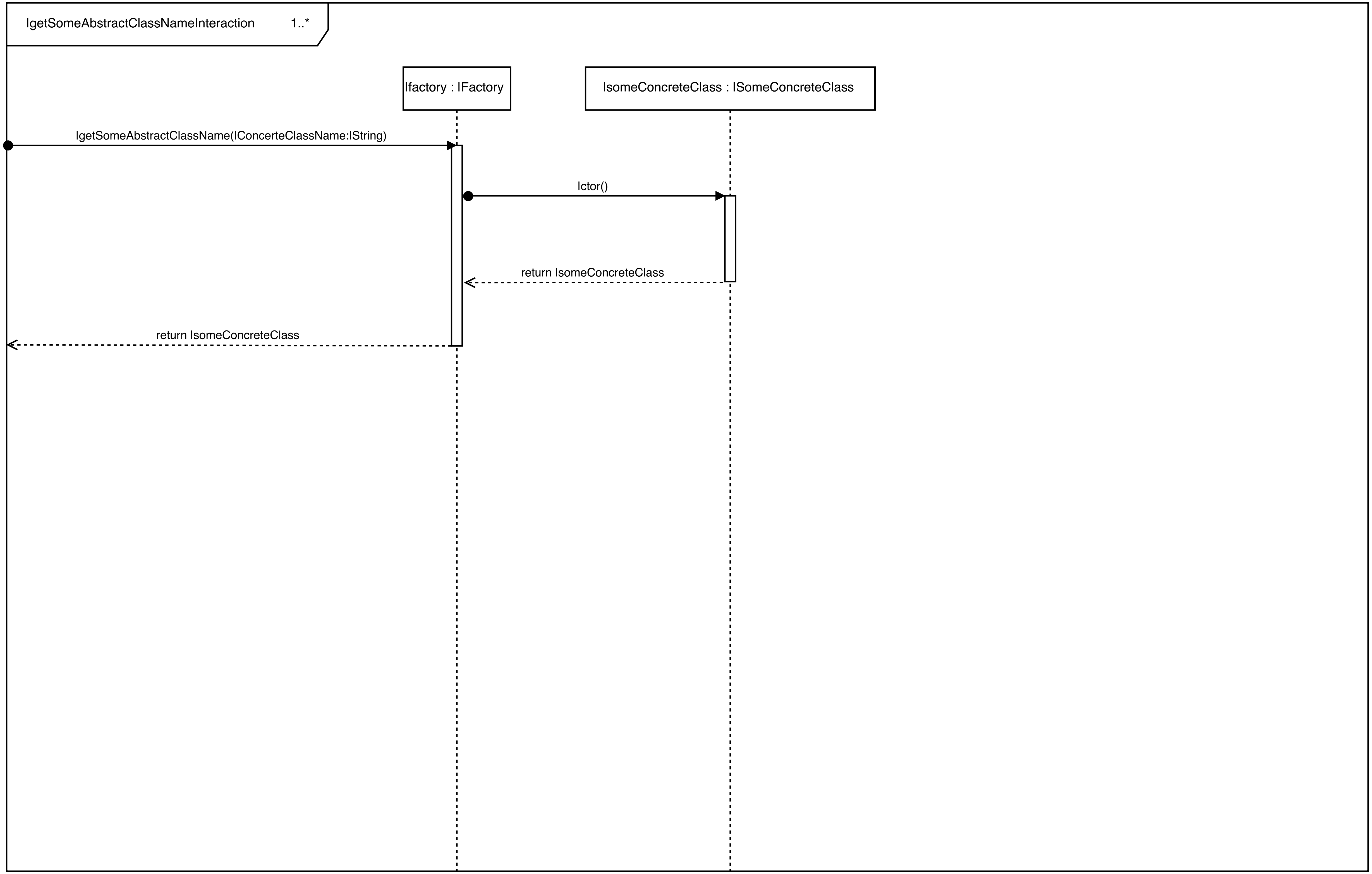
```
--
model FPS

abstract class Player
  attributes
    state: PlayerState
    panicState: PanicState
    neutralState: NeutralState
    attackState: AttackState
    dieState: DieState
    strength: Integer
    health: Integer
  operations
    init(s:Integer, h:Integer)
      begin
        self.panicState := new PanicState;
        self.neutralState := new NeutralState;
        self.attackState := new AttackState;
        self.dieState := new DieState;
        self.state := self.neutralState;
        self.strength := s;
        self.health := h;
      end
    seeOpponent(p:Player)
      begin
        self.state.seeOpponent(p);
      end
    lostOpponent()
      begin
        self.state.lostOpponent();
      end
    recieveBlow(damage:Integer)
      begin
        self.state.recieveBlow(damage);
      end

    statemachines

      psm Usage
        states
          startUp:initial
          -- Initial state after startup
          neutralState [state = neutralState]

          attackState [state = attackState]

          panicState [state = panicState]

          dieState [state = dieState]

        transitions

          startUp -> neutralState { create }
          neutralState -> attackState { [p.strength <= strength and state = neutralState]
          seeOpponent() }

          neutralState -> panicState { [p.strength > strength and state = neutralState]
          seeOpponent() }

          neutralState -> neutralState { [state = neutralState] lostOpponent() }

          neutralState -> neutralState { [state = neutralState] recieveBlow() }

          attackState -> neutralState { [state = attackState] lostOpponent() }

          attackState -> dieState { [state = attackState] recieveBlow() [health <= 0]}

          attackState -> attackState { [state = attackState] recieveBlow() [health > 0]}

          attackState -> attackState { [state = attackState] seeOpponent()}

          panicState -> neutralState { [state = panicState] lostOpponent() }

          panicState -> dieState { [state = panicState] recieveBlow() [health <= 0]}

          panicState -> panicState { [state = panicState] recieveBlow() [health > 0]}

          panicState -> panicState { [state = panicState] seeOpponent() }
      end

end

abstract class PlayerState
  attributes
  operations
    seeOpponent(p:Player)
      begin
      end
    lostOpponent()
      begin
      end
    recieveBlow(damage:Integer)
      begin
      end

end

class NeutralState < PlayerState
  attributes
  operations
    seeOpponent(p:Player)
      begin
        if p.strength > self.player.strength then
          self.player.state := self.player.panicState;
        else
```

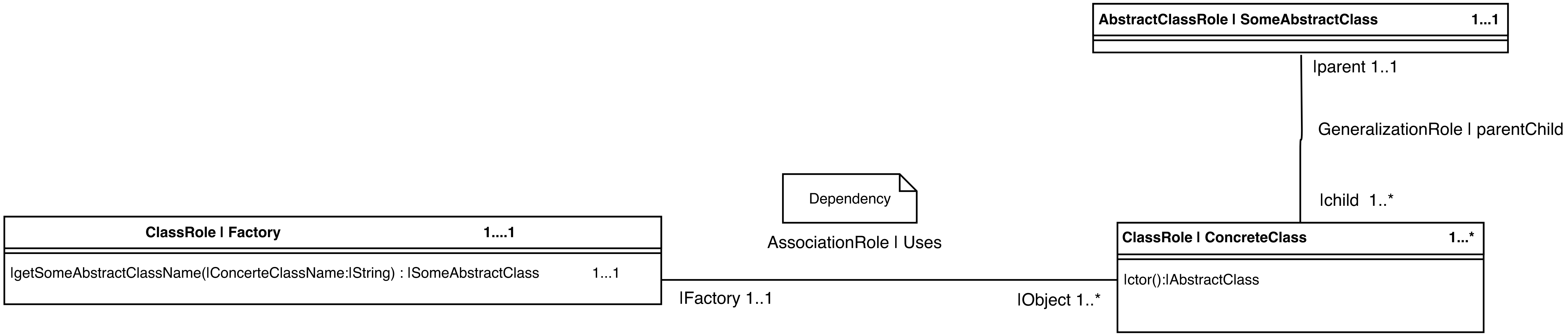
```
          self.player.state := self.player.attackState;
        end
      end
end

class AttackState < PlayerState
  attributes
  operations
    recieveBlow(damage:Integer)
      begin
        self.player.health := self.player.health - damage;
        if self.player.health <= 0 then
          self.player.state := self.player.dieState;
        end

      end
    lostOpponent()
      begin
        self.player.state := self.player.neutralState
      end
end

class DieState < PlayerState
  attributes
  operations
end

class PanicState < PlayerState
  attributes
  operations
    recieveBlow(damage:Integer)
      begin
        self.player.health := self.player.health - damage;
        if self.player.health <= 0 then
          self.player.state := self.player.dieState;
        end
      end
    lostOpponent()
      begin
        self.player.state := self.player.neutralState
      end
end

class PlayableCharacter < Player
  attributes
  operations
end

class NonPlayableCharacter < Player
  attributes
  operations
end
```

```
class WeaponsCache
  attributes
  operations
end

class Weapon
  attributes
  operations
end

class Level
  attributes

  operations
    play()
      begin
      end
end

class Ammo
  attributes
  operations
end

class Game
  attributes
  operations
    playLevel()
      begin
      end
end

association weapons between
  WeaponsCache[1] role weaponsCache
  Weapon[0..*] role weapons
end

association ammo between
  Weapon[1] role weapon
  Ammo[0..*] role ammo
end

association playerWeapons between
  Player[1] role player
  Weapon[0..2] role weapons
end

association levelPlayers between
  Level[1] role level
  Player[0..*] role players
end
```

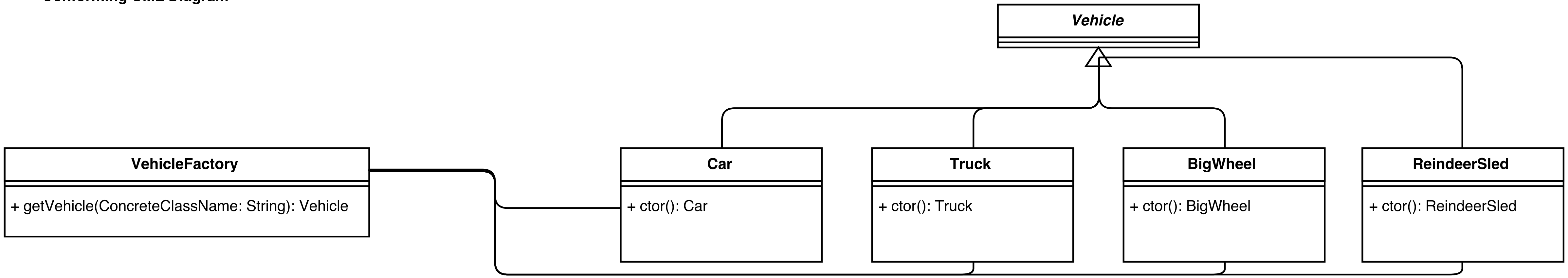
```
association levelCaches between
  Level[1] role level
  WeaponsCache[0..*] role weaponsCache
end

association cacheAmmo between
  WeaponsCache[1] role weaponsCache
  Ammo[0..*] role ammo
end

association gameLevel between
  Game[1] role game
  Level[1] role level
end

association playerState between
  Player[1] role player
  PlayerState[1] role state
end
```

```
!create human:PlayableCharacter
!human.init(100,100)
!insert (human, human.neutralState) into playerState
!insert (human, human.attackState) into playerState
!insert (human, human.panicState) into playerState
!insert (human, human.dieState) into playerState

!create bot:NonPlayableCharacter
!bot.init(100,100)
!insert (bot, bot.neutralState) into playerState
!insert (bot, bot.attackState) into playerState
!insert (bot, bot.panicState) into playerState
!insert (bot, bot.dieState) into playerState
```


lgetSomeAbstractClassNameInteraction
1..*
lfactory : IFactory
lsomeConcreteClass : lSomeConcreteClass
lgetSomeAbstractClassName(lConcerteClassName:lString)
lctor()
return lsomeConcreteClass
return lsomeConcreteClass

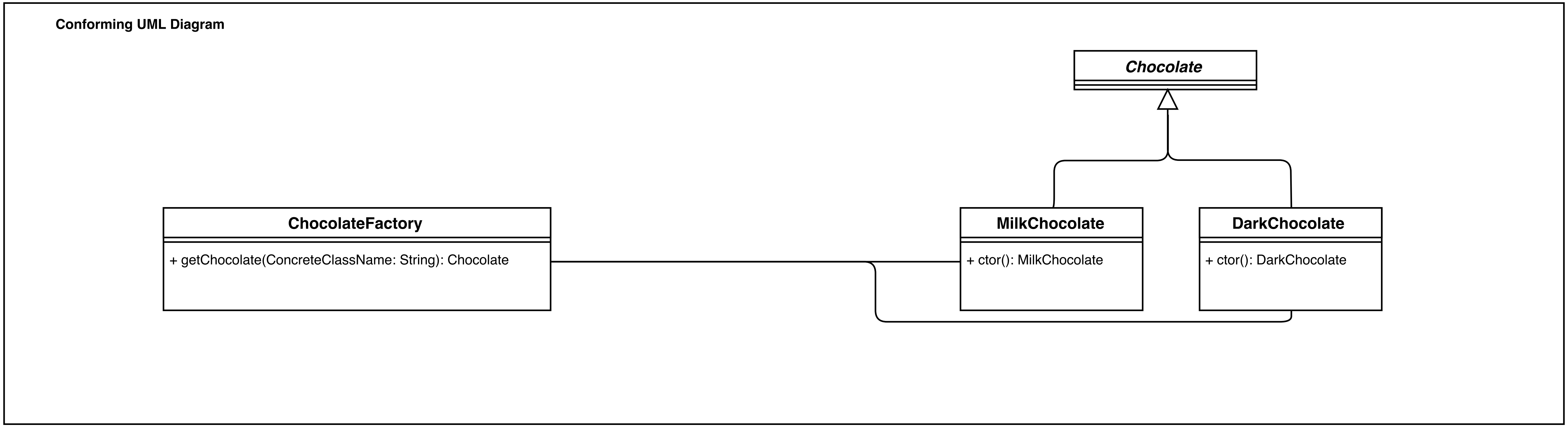

AbstractClassRole | SomeAbstractClass 1...1
|parent 1..1
GeneralizationRole | parentChild
|child 1..*
Dependency
AssociationRole | Uses
ClassRole | Factory 1....1
|getSomeAbstractClassName(|ConcerteClassName:|String) : |SomeAbstractClass 1...1
|Factory 1..1
|Object 1..*
ClassRole | ConcreteClass 1...*
|ctor():|AbstractClass

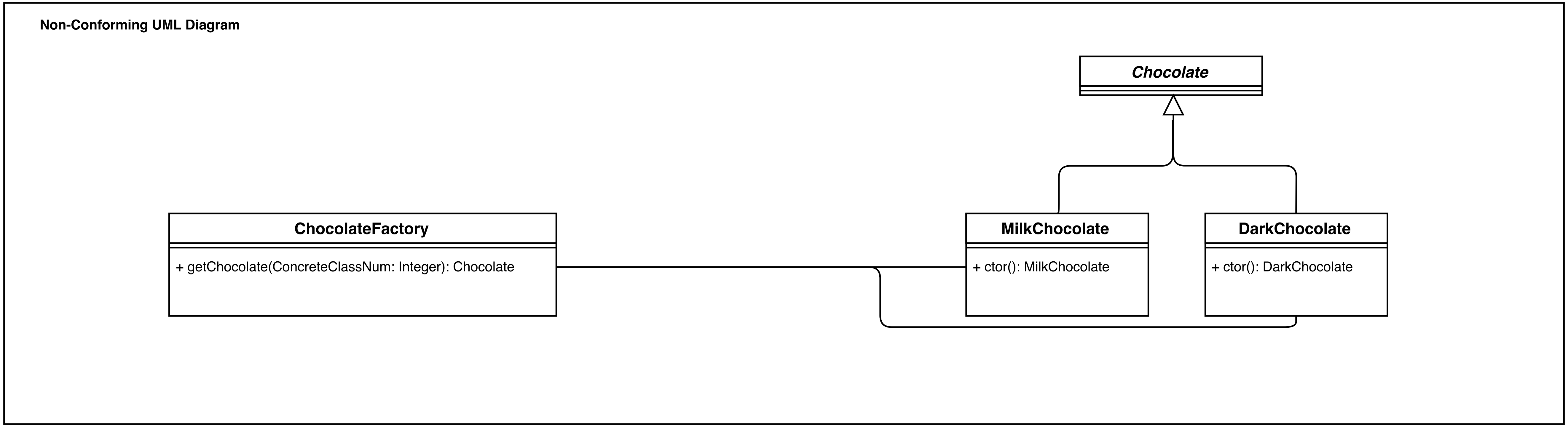
Conforming UML Diagram



iii) How would you achieve balance between a restrictive vs. a lenient metamodel? Why is this an important issue?

We would achieve balance between a restrictive and lenient metamodel by examining the project which we are applying our RBML meta-model to and loosening or tightening constraints dependent upon the objectives of the project. This balance would be achieved by changing how general the constraints are made. In the Factory Method Pattern we have restricted instances of UML which match to those who have a function for getSomeAbstractClass who's only parameter is a String. We could generalize this to be a GenericKey of some kind, but have decided we would narrow the scope of the Factory Method Pattern we are matching. This is an important issue because being too lenient could result in code which becomes harder to manage and extend, while being too restrictive could defeat the polymorphism inherent in Design Patterns.


Conforming UML Diagram


Non-Conforming UML Diagram