Object Calesthenics

Findings

Eckhard Grodtke CSS-Insurance

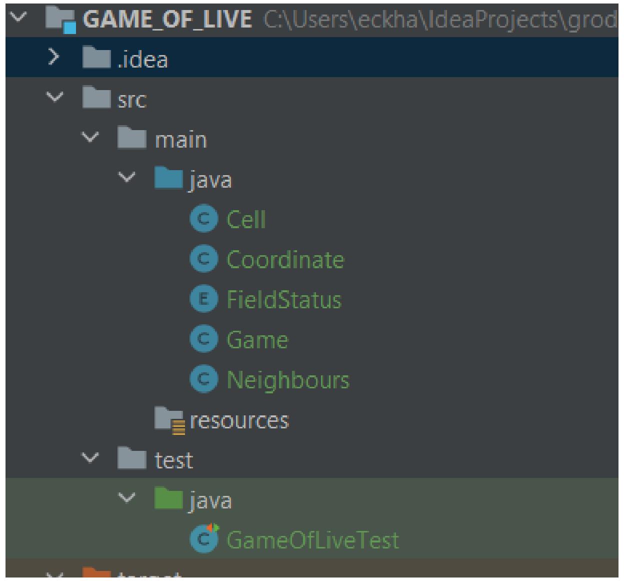
- Only One Level Of Indentation Per Method
- Don’t Use The ELSE Keyword
- Wrap All Primitives And Strings
- All classes must have state
- First Class Collections
- One Dot Per Line
- No Getters no Setters only Private Fields
- Don’t Abbreviate
- Keep All Entities Small
- No Classes With More Than Two Instance Variables

Review first iteration Game of Live Kata

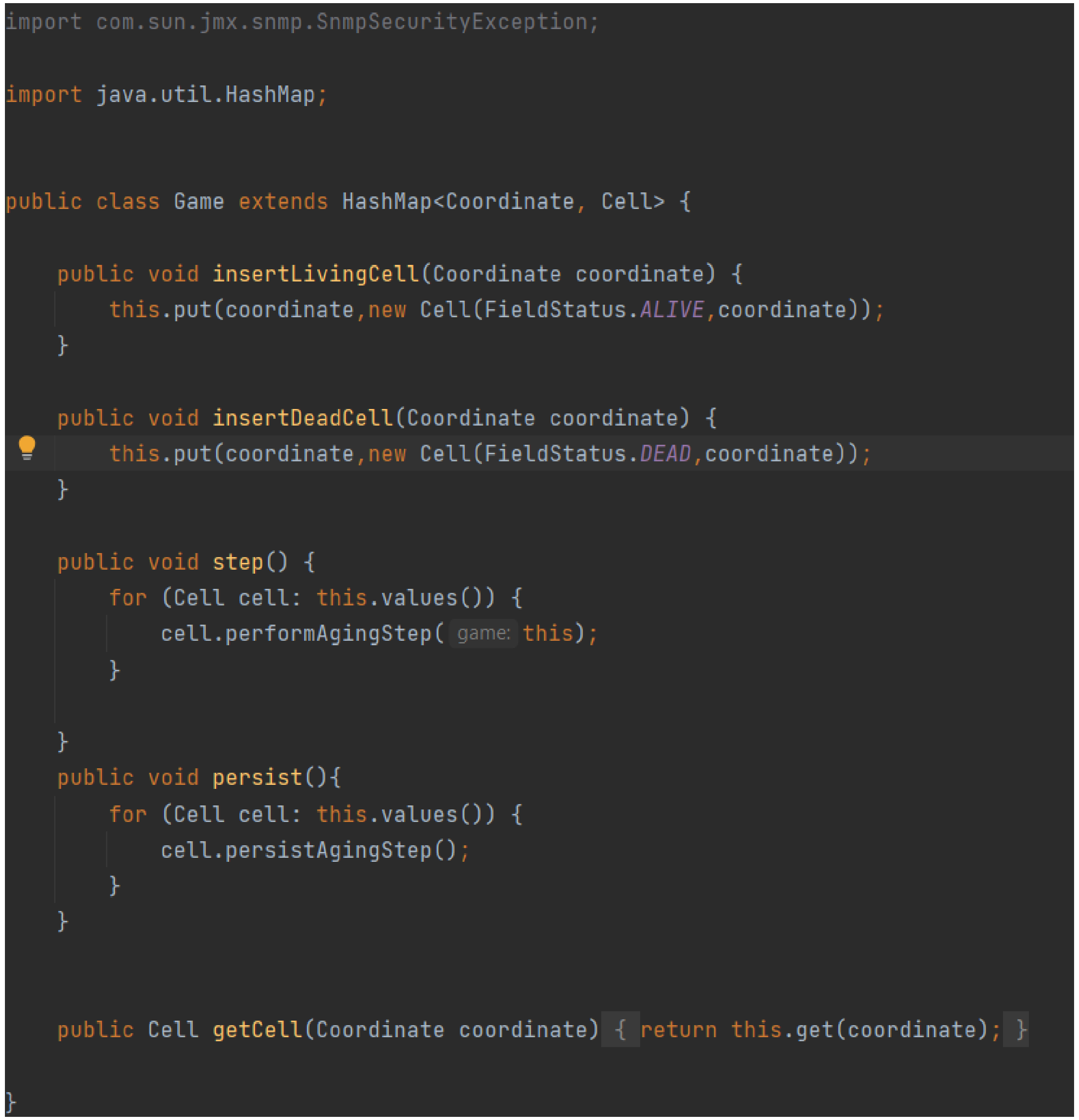
Rules:

```
/*Any live cell with fewer than two live neighbours dies, as if caused by under-population.
  Any live cell with two or three live neighbours should live on to the next generation.
  Any live cell with more than three live neighbours dies, as if by over-population.
  Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction
 */
```

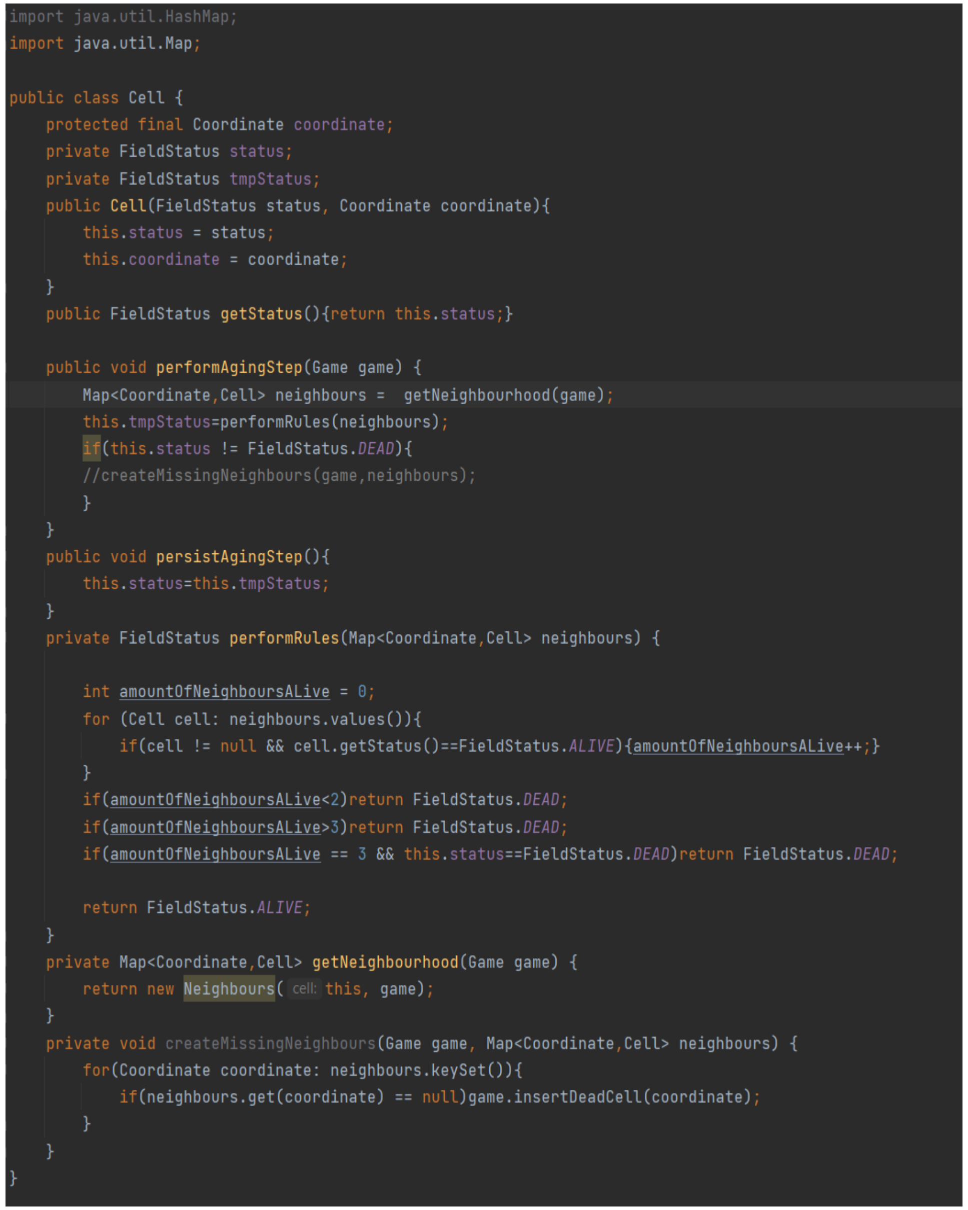
Object view



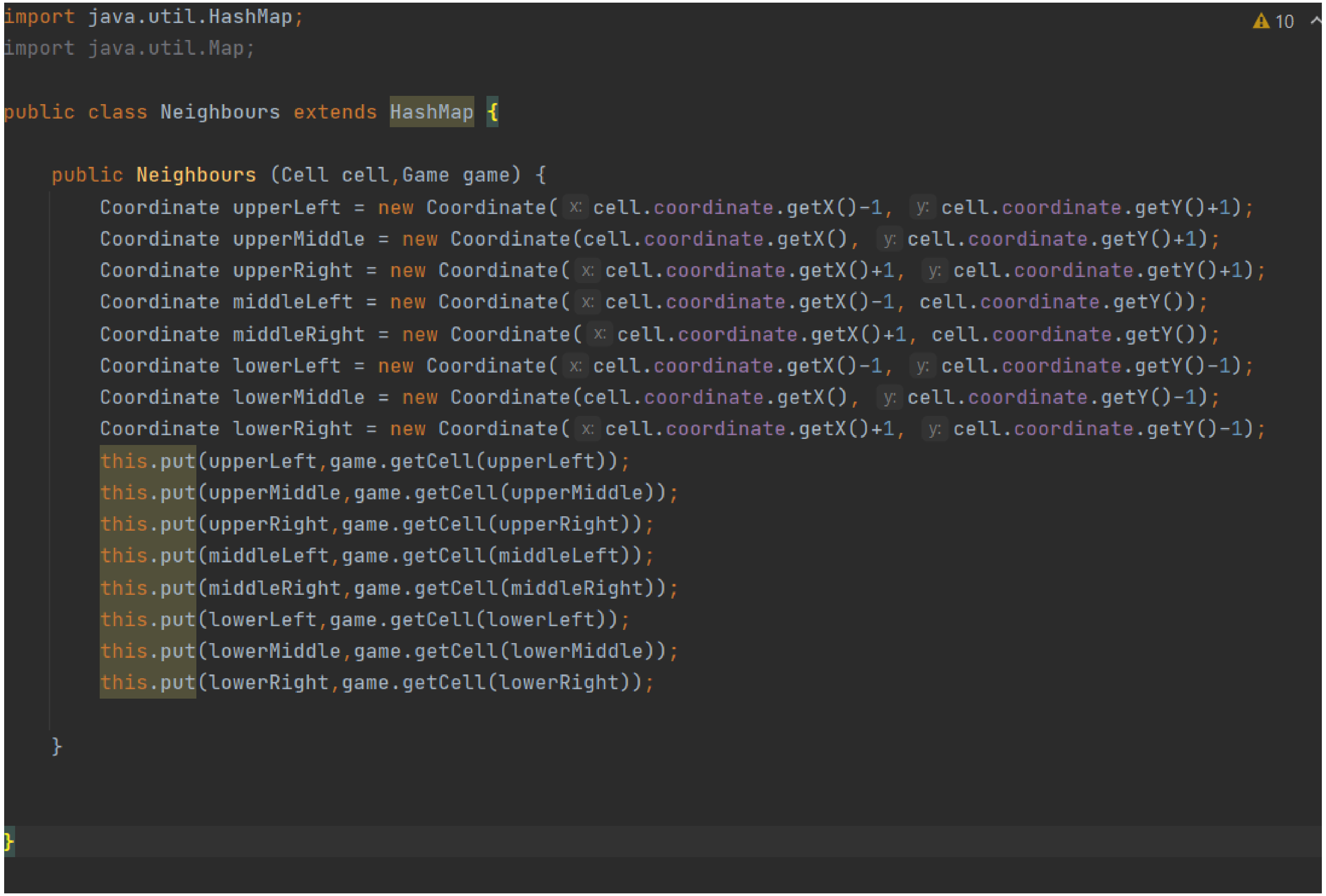
```
GAME_OF_LIVE C:\Users\eckha\IdeaProjects\grod
  .idea
  src
    main
      java
        Cell
        Coordinate
        FieldStatus
        Game
        Neighbours
      resources
    test
      java
        GameOfLiveTest
```



```java
import com.sun.jmx.snmp.SnmpSecurityException;

import java.util.HashMap;


public class Game extends HashMap<Coordinate, Cell> {

    public void insertLivingCell(Coordinate coordinate) {
        this.put(coordinate,new Cell(FieldStatus.ALIVE,coordinate));
    }

    public void insertDeadCell(Coordinate coordinate) {
        this.put(coordinate,new Cell(FieldStatus.DEAD,coordinate));
    }

    public void step() {
        for (Cell cell: this.values()) {
            cell.performAgingStep( game: this);
        }

    }
    public void persist(){
        for (Cell cell: this.values()) {
            cell.persistAgingStep();
        }
    }


    public Cell getCell(Coordinate coordinate) { return this.get(coordinate); }

}
```



```java
import java.util.HashMap;
import java.util.Map;

public class Cell {
    protected final Coordinate coordinate;
    private FieldStatus status;
    private FieldStatus tmpStatus;
    public Cell(FieldStatus status, Coordinate coordinate){
        this.status = status;
        this.coordinate = coordinate;
    }
    public FieldStatus getStatus(){return this.status;}

    public void performAgingStep(Game game) {
        Map<Coordinate,Cell> neighbours =  getNeighbourhood(game);
        this.tmpStatus=performRules(neighbours);
        if(this.status != FieldStatus.DEAD){
        //createMissingNeighbours(game,neighbours);
        }
    }
    public void persistAgingStep(){
        this.status=this.tmpStatus;
    }
    private FieldStatus performRules(Map<Coordinate,Cell> neighbours) {

        int amountOfNeighboursALive = 0;
        for (Cell cell: neighbours.values()){
            if(cell != null && cell.getStatus()==FieldStatus.ALIVE){amountOfNeighboursALive++;}
        }
        if(amountOfNeighboursALive<2)return FieldStatus.DEAD;
        if(amountOfNeighboursALive>3)return FieldStatus.DEAD;
        if(amountOfNeighboursALive == 3 && this.status==FieldStatus.DEAD)return FieldStatus.DEAD;

        return FieldStatus.ALIVE;
    }
    private Map<Coordinate,Cell> getNeighbourhood(Game game) {
        return new Neighbours( cell: this, game);
    }
    private void createMissingNeighbours(Game game, Map<Coordinate,Cell> neighbours) {
        for(Coordinate coordinate: neighbours.keySet()){
            if(neighbours.get(coordinate) == null)game.insertDeadCell(coordinate);
        }
    }
}
```



```java
import java.util.HashMap;
import java.util.Map;

public class Neighbours extends HashMap {

    public Neighbours (Cell cell,Game game) {
        Coordinate upperLeft = new Coordinate( x: cell.coordinate.getX()-1,  y: cell.coordinate.getY()+1);
        Coordinate upperMiddle = new Coordinate(cell.coordinate.getX(),  y: cell.coordinate.getY()+1);
        Coordinate upperRight = new Coordinate( x: cell.coordinate.getX()+1,  y: cell.coordinate.getY()+1);
        Coordinate middleLeft = new Coordinate( x: cell.coordinate.getX()-1, cell.coordinate.getY());
        Coordinate middleRight = new Coordinate( x: cell.coordinate.getX()+1, cell.coordinate.getY());
        Coordinate lowerLeft = new Coordinate( x: cell.coordinate.getX()-1,  y: cell.coordinate.getY()-1);
        Coordinate lowerMiddle = new Coordinate(cell.coordinate.getX(),  y: cell.coordinate.getY()-1);
        Coordinate lowerRight = new Coordinate( x: cell.coordinate.getX()+1,  y: cell.coordinate.getY()-1);
        this.put(upperLeft,game.getCell(upperLeft));
        this.put(upperMiddle,game.getCell(upperMiddle));
        this.put(upperRight,game.getCell(upperRight));
        this.put(middleLeft,game.getCell(middleLeft));
        this.put(middleRight,game.getCell(middleRight));
        this.put(lowerLeft,game.getCell(lowerLeft));
        this.put(lowerMiddle,game.getCell(lowerMiddle));
        this.put(lowerRight,game.getCell(lowerRight));

    }


}
```

Solutions found or used to fulfill Object Calesthenics requirements

- Put parent object into method call
- Overwrite equals and hash
- Selfinitializing object
- Inheritance
- Garbage collection as Part of Implementaion

Findings:

All Rules of Object Calesthenics are maximum Requiremets for use in a practice.

In real live we should try to approach to the specification, but the size of the project and the complexity of professional requirements make compromises necessary.

Practice Object Calesthenics can open our mind. But even with open minds we are able to produce monsters.