

EclipseUML by examples

Università degli Studi di Bologna

Facoltà di Scienze MM. FF. NN.

Corso di Laurea Triennale in Scienze di Internet

Anno Accademico 2004-2005

Laboratorio di Sistemi e Processi Organizzativi



Module Road Map



1. **How to Install**
2. **Class Diagram Example**
3. **The Game of Life**

Module Road Map



1. Overview and how to install

2. Class Diagram Example
3. The Game of Life

What is Omondo EclipseUML?



Omondo EclipseUML is a visual modeling tool

natively integrated with Eclipse

<http://www.eclipseuml.com>

one of the most successful Eclipse plug-ins with more than 250,000 downloads in the last 12 months

the leading UML Eclipse plugin

Omondo offers two different tools:

EclipseUML Free Edition

which is a UML specialized tool;

EclipseUML Enterprise Edition

which provides data modeling, UML, J2ee, and business process modeling,

time limited trial



How to install Eclipse



<http://www.eclipseuml.com/download/free/index.jsp>

download Eclipse 2.1.1 in accordance with your OS

eclipse-SDK-2.1.1-win32.zip or

eclipse-SDK-2.1.1-linux-gtk.zip

download EclipseUML latest version

eclipseuml-installer_1.2.1.20031103.jar

according to your OS, unzip eclipse-SDK into

C:\eclipse

\usr\local\eclipse

run

java -jar eclipseuml-installer_1.2.1.20031103.jar

specifying the right installation directory

Module Road Map



1. Overview and how to install

2. Class Diagram Example

3. The Game of Life

Class Diagram Example

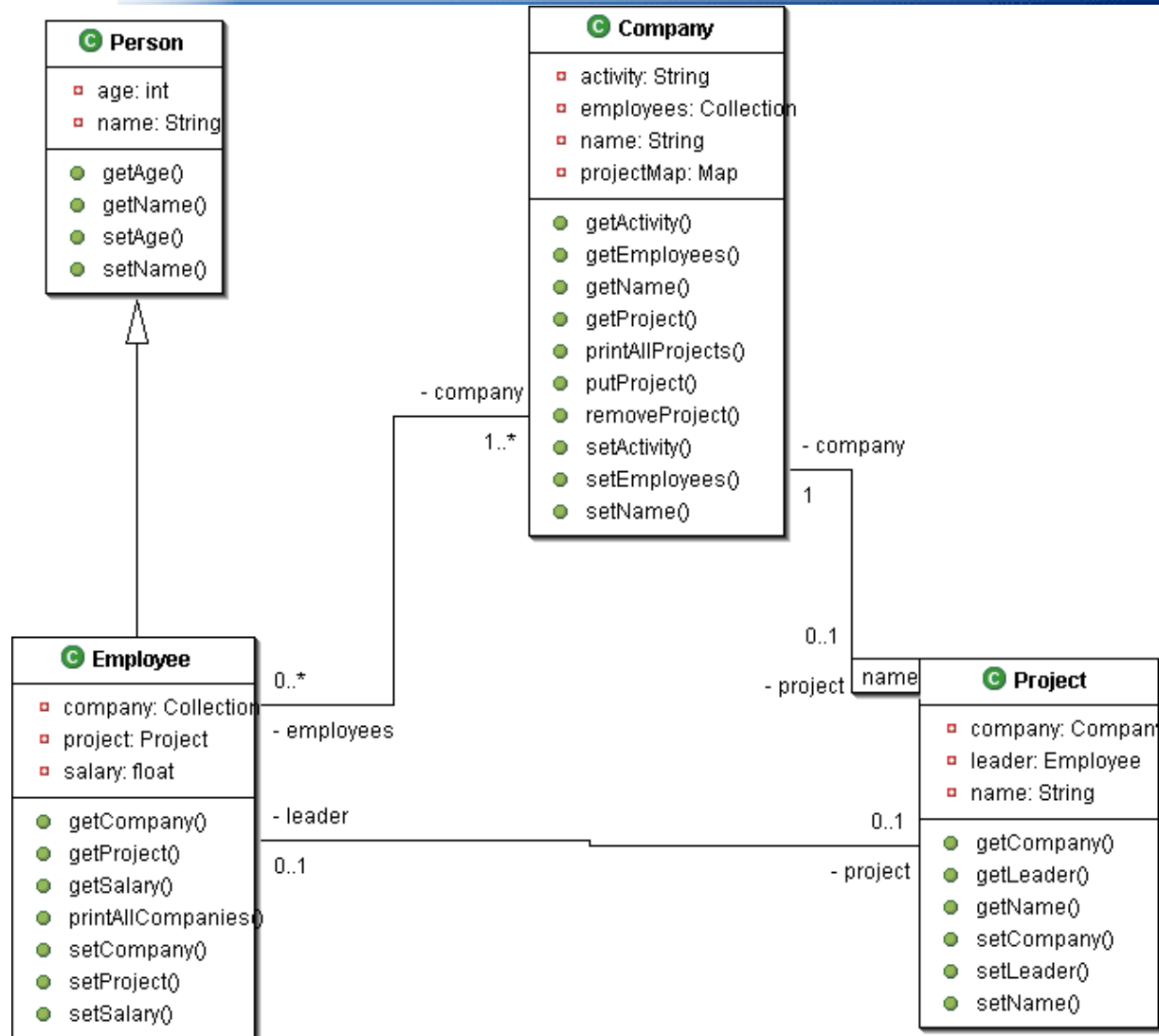


The following path from the eclipse directory
eclipse\plugins\com.omondo.uml.doc.user_1.2.1\doc\diagrams\classDiagram\ClassDiagramExample\ClassDiagramExample.html

shows a class diagram creation.

This is a good point to start.

Class Diagram Example



Module Road Map



1. Overview and how to install
2. Class Diagram Example
- 3. The Game of Life**

References



The following slides are extracted from a courseware which is part of the Open Source “ECESIS” project

The courseware is produced by, and is copyrighted by Espirity, Inc. and CMA

To get the whole course material go to

<http://www.eclipse.org/ecesis/>

Rules of the Game of Life



Life is played on a grid of square cells (like a chess board) but extending infinitely in every direction.

A cell can be **live** or **dead**.

A live cell is shown by putting a marker on its square.

A dead cell is shown by leaving the square empty.

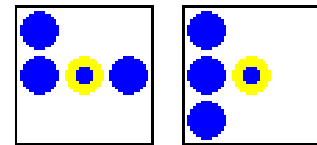
Each cell in the grid has a neighborhood consisting of the eight cells in every direction including diagonals.

To apply one step of the rules, we count the number of live neighbors for each cell. What happens next depends on this number.

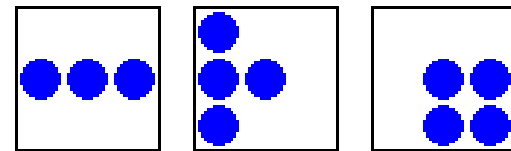
Rules of the Game of Life



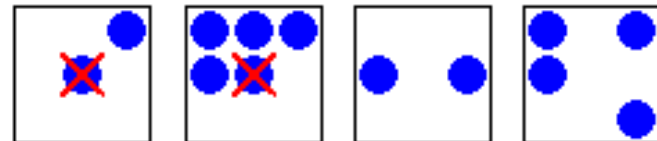
A dead cell with exactly three live neighbors becomes a live cell (**birth**).



A live cell with two or three live neighbors stays alive (**survival**).



In all other cases, a cell dies or



What objects can we identify?



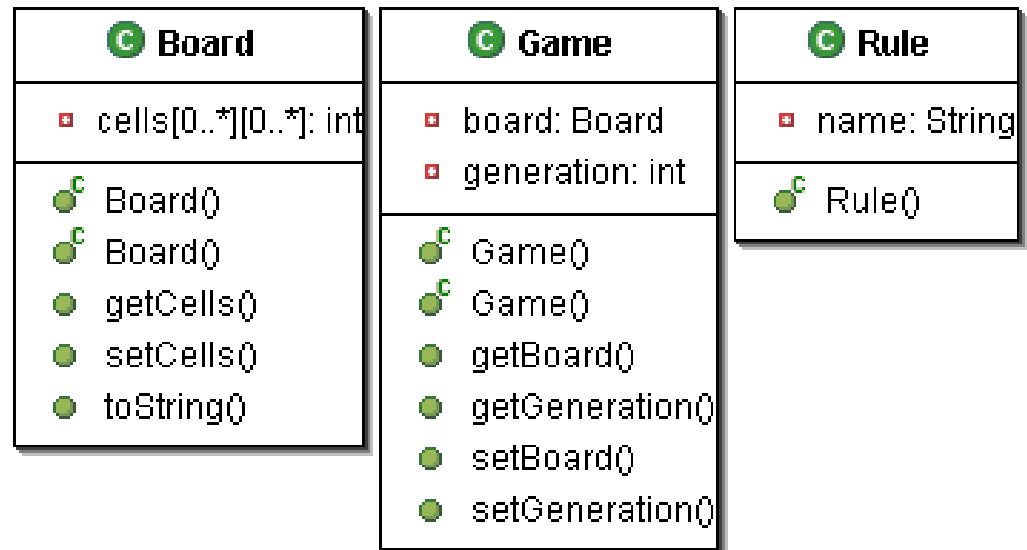
The basic classes in the system may be

Board

Game

Rule

Create 3 classes



Board.java



Edit Board.java *constructors* and *toString*:

```
public Board() {
    this.cells = new int[10][10];
}
public Board(int[][] cells) {
    this.cells = cells;
}
public String toString() {
    return "Board for game of life";
}
```

Game.java and Rule.java



Edit Game.java *construtors*

```
public Game(Board aBoard) {
    this.board = aBoard;
    this.generation = 1;
}

public Game() {
    setBoard(new Board());
}
```

Edit Rule.java *construtor*

```
public Rule(String name) {
    this.name = name;
}
```

Use scrapbook pages



Inspect with scrapbook the following code:

```
int[][] boardCells = {
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 1}};
Board board = new Board(boardCells);
Game game = new Game(board);
game
```


Singleton pattern



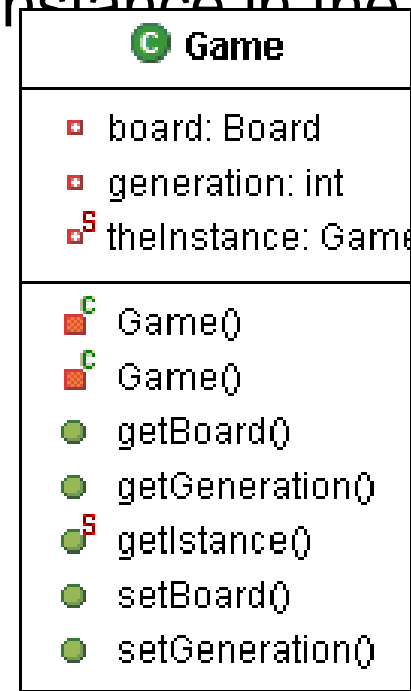
How many instances of the Game class will we need at the same time in one single application?

Just one.

Define a private static field called `theInstance` in the Game class.

Lazy initialization:

```
public static Game getInstance() {  
    if (theInstance == null)  
        theInstance = new Game();  
  
    return theInstance;  
}
```



Edit toString



Edit toString method in Board class:

```
public String toString(){
    StringBuffer buffer = new StringBuffer();
    for (int i = 0; i < getCells().length; i++){
        buffer.append("\n");
        for (int j = 0; j < getCells()[0].length; j++){
            if (getCells()[i][j] == 0)
                buffer.append("0");
            else
                buffer.append("1");
        }
    }
    return buffer.toString();
}
```

Use scrapbook pages



Execute with scrapbook the following code:

```
int[][] boardCells = {
    {0, 0, 0, 1, 0, 0, 0 , 0, 0, 1},
    {0, 0, 0, 1, 0, 0, 0 , 0, 0, 1},
    {0, 0, 1, 1, 1, 0, 0 , 0, 0, 1},
    {0, 0, 1, 1, 1, 0, 0 , 0, 0, 1},
    {0, 0, 1, 1, 1, 0, 0 , 0, 0, 1},
    {0, 0, 0, 1, 0, 0, 0 , 0, 0, 1},
    {0, 0, 0, 0, 0, 0, 0 , 0, 0, 1},
    {0, 0, 0, 1, 0, 0, 0 , 0, 0, 1},
    {0, 0, 1, 1, 1, 0, 0 , 0, 0, 1},
    {0, 0, 1, 1, 1, 0, 0 , 0, 0, 1}};
Board board = new Board(boardCells);
System.out.println(board)
```

Board class Refactoring

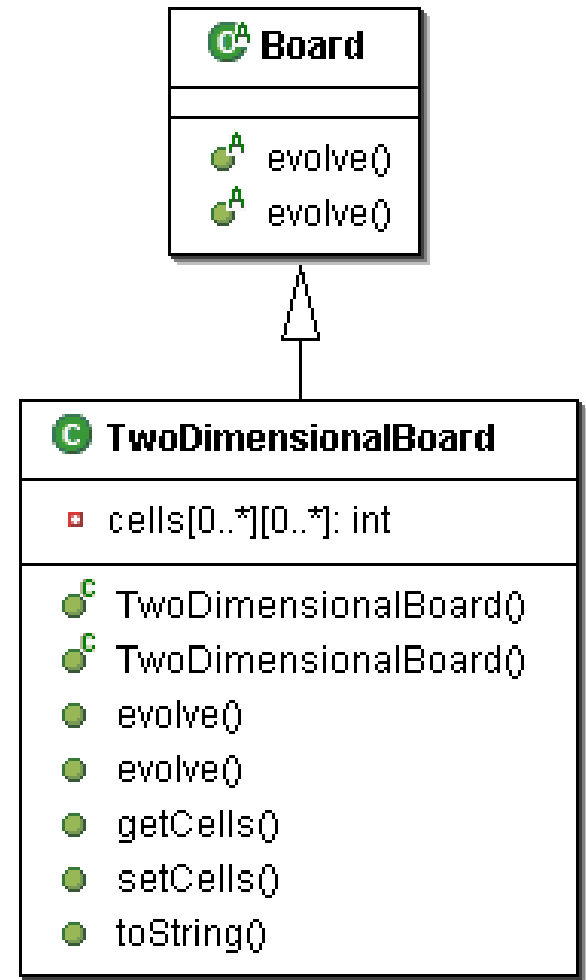


Change the Board class name to
TwoDimensionalBoard
Refactor->Rename

Create an abstract class Board with
abstract evolve methods

```
public abstract void evolve  
(Game);
```

```
public abstract void evolve  
(Game, int);
```

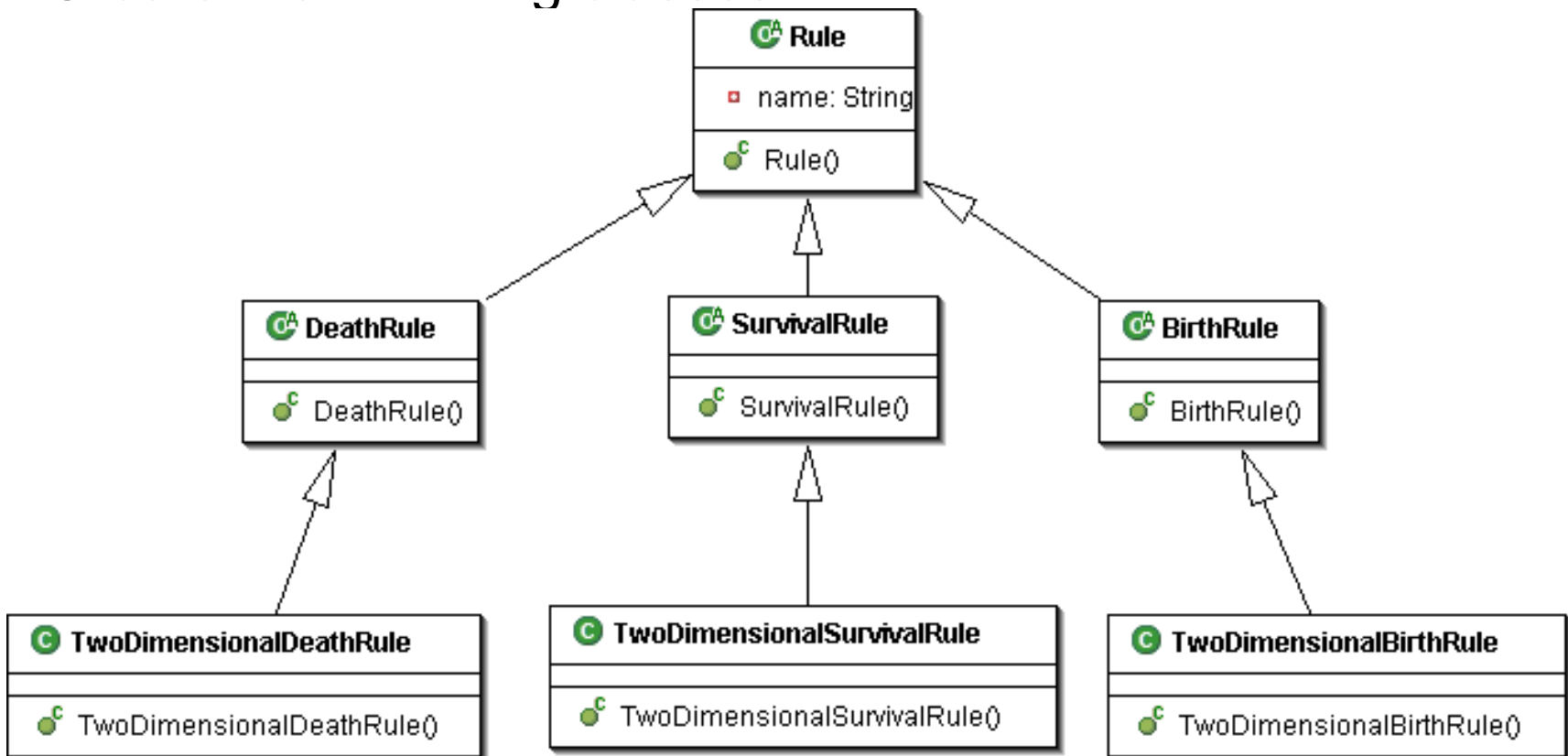


Rule class Refactoring



Change the `Rule` class definition to abstract

Create the following classes:



Create Interface



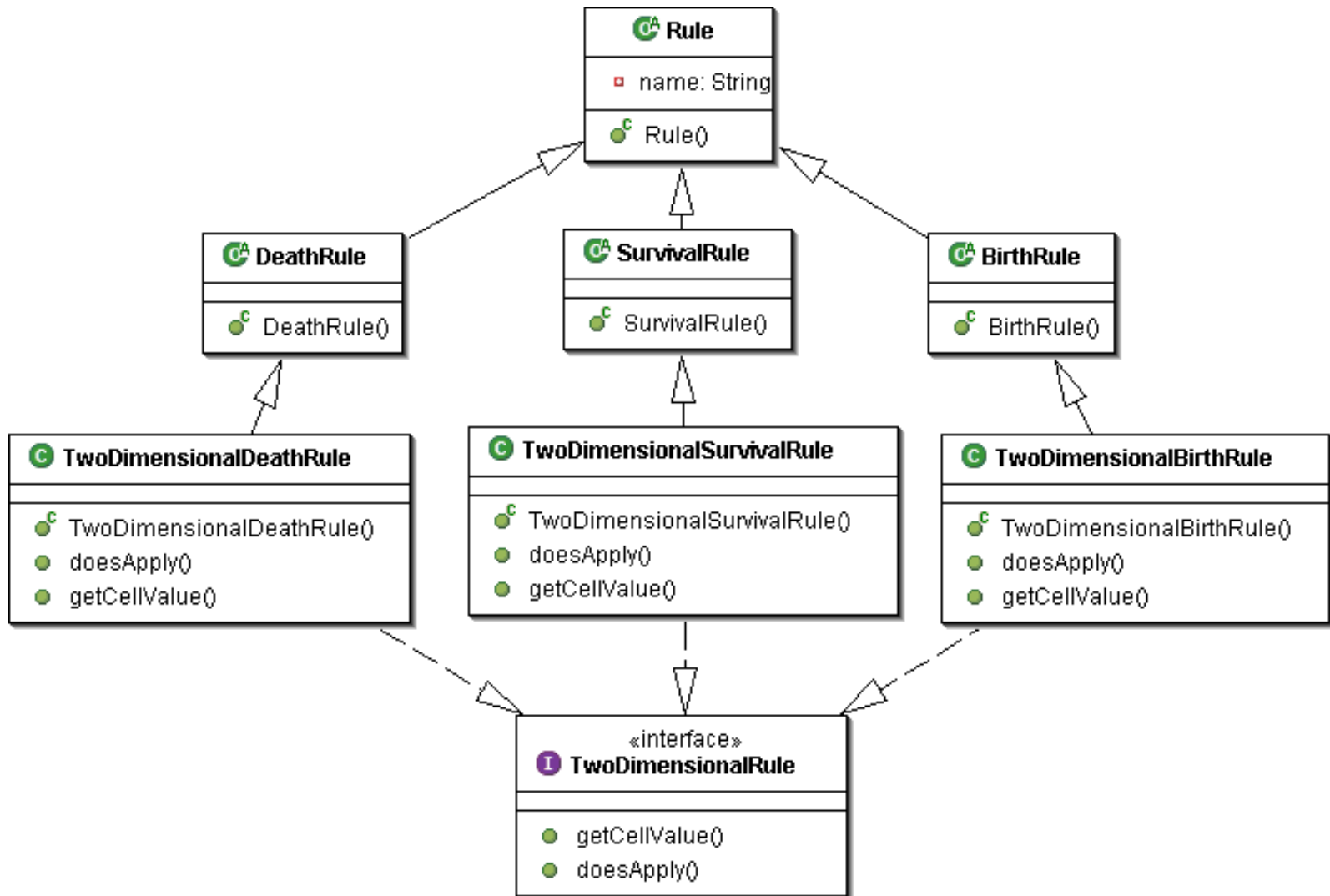
Create new interface `TwoDimensionalRule` with two methods:

`doesApply()`

`getCellValue()`

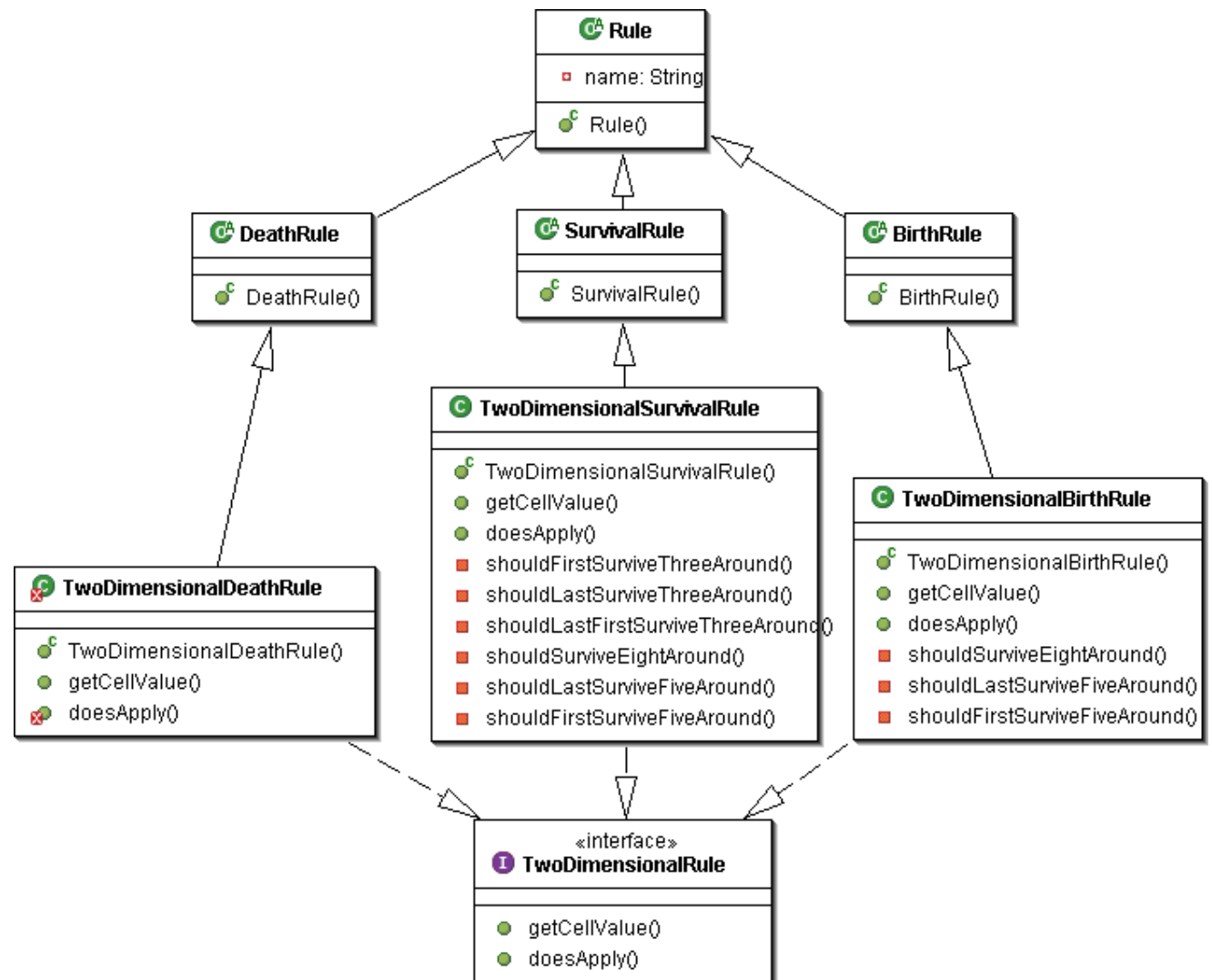
it should simply return int value,
for birth and survival rules it would be 1,
for death rule it would be 0.

Create Interface



Import rules

Import the 3 provided rule classes.



Redesign the Game class



Make the Game class previously defined abstract.

Create TwoDimensionalGame class that inherits from Game class.

Default constructor in the Game class should initialize rules:

```
public Game() {  
    this.rules = new Rule[3];  
    this.generation = 1;  
}
```

Move singleton implementation to TwoDimensionalGame and let the following be the constructor of the class:

```
private TwoDimensionalGame(){  
    setBoard(new TwoDimensionalBoard());  
    getRules()[0] = new TwoDimensionalBirthRule("Birth Rule");  
    getRules()[1] = new TwoDimensionalSurvivalRule("Survival Rule");  
    getRules()[2] = new TwoDimensionalDeathRule("Death Rule");  
}
```

The evolve method



Implement evolve method in the TwoDimensionalBoard class.

```
public void evolve(Game aGame) {
    boolean doesAnyApply = false;
    TwoDimensionalRule rule = null;
    for (int i = 0; i < cells.length; i++){
        for (int j = 0; j < cells[0].length; j++){
            doesAnyApply = false;
            for (int k = 0; k < aGame.getRules().length; k++){
                rule = (TwoDimensionalRule)aGame.getRules()[k];
                if(rule.doesApply(getCells(), i, j)){
                    getCells()[i][j] = rule.getCellValue();
                    doesAnyApply = true;
                    break;
                }
            }
        }
        if (!(doesAnyApply))
            System.out.println("No rules apply for cell[" + i + "][" + j + "]);
    }
}
```

Evolve and run methods



Implement a second `evolve` method which evolves board as many times as passed `index`:

```
public void evolve(Game aGame, int index){
    if (index == 0) return;
    for (int i = 1; i <= index; i++){
        evolve(aGame);
        System.out.println(this);
    }
}
```

Implement `run` method in the `Game` class:

```
public void run(){
    getBoard().evolve(this, getGeneration());
}
```

Test the game



Create a new package and a new class with main method like the following:

```
public static void main(String[] args) {
    int[][] boardCells = {
        {0, 0, 0, 1, 0, 0, 1, 0, 1, 0},
        {0, 0, 1, 0, 1, 0, 1, 0, 0, 0},
        {1, 0, 0, 1, 0, 0, 1, 0, 0, 0},
        {0, 0, 0, 1, 0, 0, 1, 0, 0, 0},
        {0, 0, 1, 0, 1, 0, 1, 1, 0, 0},
        {0, 0, 0, 1, 0, 0, 1, 1, 1, 0},
        {0, 0, 0, 1, 0, 0, 1, 1, 0, 1},
        {1, 0, 1, 0, 1, 0, 1, 0, 1, 0},
        {0, 0, 1, 0, 1, 0, 1, 0, 1, 1},
        {0, 0, 0, 1, 0, 0, 1, 0, 0, 1}};
    TwoDimensionalBoard board = new TwoDimensionalBoard(boardCells);
    TwoDimensionalGame game = TwoDimensionalGame.getInstance();
    game.setBoard(board);
    game.setGeneration(10);
    game.run();
}
```