Start Assignment

Due Monday by 11:59pm **Points** 40 **Submitting** a file upload **File Types** zip
Available Mar 10 at 12am - Apr 12 at 11:59pm about 1 month

Unit 5: Polymorphism, Interfaces, and Exception Handling **13 of 14**

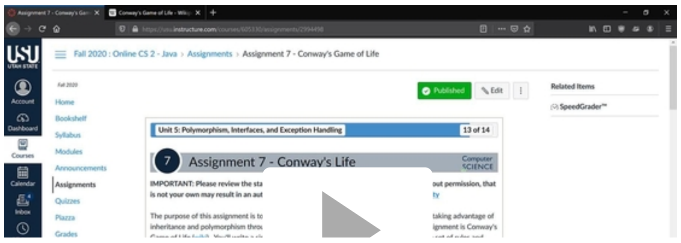
7 Assignment 7 - Conway's Life

Computer SCIENCE

IMPORTANT: Please review the statement on Academic Integrity. Submitting code, without permission, that is not your own may result in an automatic F for the class: Statement on Academic Integrity

The purpose of this assignment is to give you more experience in developing Java classes, taking advantage of inheritance and polymorphism through inheritance. The subject you'll be using for this assignment is Conway's Game of Life (wiki (https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life)). You'll write a simulator that updates the state of a word according to a set of rules and then render that state to the console, animating it over time. I think you'll enjoy this, it is fun to play around with once it is up and going.

Watch the following video for a detailed discussion on important details for this assignment...


Assignment 7 - Conway's Gam
Conway's Game of Life - Wiki
Fall 2020 : Online CS 2 - Java › Assignments › Assignment 7 - Conway's Game of Life
Fall 2020
Account
Dashboard
Courses
Calendar
Inbox
Home
Bookshelf
Syllabus
Modules
Announcements
Assignments
Quizzes
Piazza
Grades
Published
Edit
Related Items
SpeedGrader™
Unit 5: Polymorphism, Interfaces, and Exception Handling
13 of 14
7
Assignment 7 - Conway's Life
Computer SCIENCE

IMPORTANT: Please review the sta out permission, that
is not your own may result in an aut ty

The purpose of this assignment is to taking advantage of
inheritance and polymorphism throu ignment is Conway's

0:00 / 25:19 1x

Assignment

Write a program that provides an interesting animation using Conway's Game of Life rules and using at least the required patterns indicated below.

Required Patterns

- Acorn
- Block
- Blinker
- Glider

All of these patterns are shown in the wiki: **link (https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life)**

All patterns must be derived from the following class.

```
public abstract class Pattern {
    public abstract int getSizeX();
    public abstract int getSizeY();
    public abstract boolean getCell(int x, int y);
}
```

- `getSizeX` - Returns the horizontal width (in cells) of the pattern.
- `getSizeY` - Returns the vertical height (in cells) of the pattern.
- `getCell` - Returns `true` if the cell in the pattern is filled, `false` otherwise.

The derived classes may add additional private fields and methods, but no changes to the public interface allowed. You'll name these classes:

- `PatternAcorn`
- `PatternBlock`
- `PatternBlinker`
- `PatternGlider`

Simulator

Create a simulator class, named `LifeSimulator`, that can be initialized with some state (of patterns) and then update the game of life simulation. Use the following as the (required) public interface for the class.

```
public class LifeSimulator {
    public LifeSimulator(int sizeX, int sizeY) { ... }

    public int getSizeX() { ... }
    public int getSizeY() { ... }
    public boolean getCell(int x, int y) { ... }

    public void insertPattern(Pattern pattern, int startX, int startY) { ... }
    public void update() { ... }
};
```

- The constructor accepts a `sizeX` and `sizeY` indicating the size of the world. The code I have provided passes these values into the `LifeSimulator` constructor.
- `insertPattern` - Adds the pattern to the world, with the upper left corner beginning at `startX` and `startY`. Adding a pattern means copying the cells from the pattern into the internal grid representation of the simulator. You don't keep references to the original patterns, you copy the cell values into the simulator.
- `update` - Performs a single step update of the world, following the four rules specified in the wiki article. The following steps provide an overview for how to do an update:
    1. Make a new grid (array) that is the same size as the current grid
        - You'll have a reference called something like 'original' and a reference called 'updated'. Where 'original' refers to the current grid state and 'updated' refers to the new grid you just created.
    2. For each cell in the original grid, use the 4 rules to decide the state of that cell in the updated grid.
    3. Put the value of the updated cell state in the updated grid. You never change any of the cell values in the original grid, you only read from it, always writing the updated values in the updated grid.
    4. When finished going through all the cells and setting their values in the updated grid, set the reference of 'original' to refer to the 'updated' grid.

- `getSizeX` & `getSizeY` - Return the size of the world.
- `getCell` - Returns `true` if the world cell is alive, `false` otherwise.

You may add any private fields and methods you like, but can not modify the public interface in any way.

Other notes

- Cells on the edge of the field should count cells past the edge (those that don't exist) and dead cells; in other words, they don't count.
  - If you want to be cool, feel free to "wrap" and count the cells from the other edge. This is definitely not required.

Renderer

Write a `render` method in the `ConwaysLife` class that takes a `LifeSimulator` and renders it to the console (described below).

```
public static void render(LifeSimulator simulation, Screen screen, TextGraphics graphics) { ... }
```

It is best to only render cells that are alive. In other words, loop through the world and when you find a cell that is alive, use `graphics.setCharacter(...)` to draw a character at a specific location in the console. If you draw all cells, dead and alive, the rendering is slower than necessary.

Console Rendering

The provided code framework utilizes a third-party library called Lanterna for conosle/text rendering (**https://github.com/mabe02/lanterna/blob/master/docs/contents.md (https://github.com/mabe02/lanterna/blob/master/docs/contents.md)**). The code has a basic demonstration for how to use it to render text to any location on the console. The only method you'll really need to utilize is `graphics.setCharacter`, as noted above, but you can look at the provided code to see how it is all working together.

Animation

Your program will run for some number of steps for the animation for your demonstration. You can use a simple counted for loop for this, but you'll want to pause for a little bit between each animation step. You can use a line of code like the following to pause for some number of milliseconds...

```
Thread.sleep(50); // This will sleep for 50 milliseconds
```

To add a pattern to your simulator, for your animation demonstration, you'll have code something along these lines...

```
LifeSimulator mySim = new LifeSimulator(...);
mySim.insertPattern(new PatternBlock(), 0, 0);
mySim.insertPattern(new PatternBlinker(), 0, 10);
mySim.insertPattern(new PatternGlider(), 15, 15);
```

Then go into an animation loop or several loops. You can have multiple loops, with each demonstrating something different. You can insert patterns during the animation, etc. Don't feel limited to only adding patterns at the start and letting it go from there, this is your chance to be creative and have fun.

Be sure to look at the code framework I've provided, it gives a good guideline for how to update and render the simulator state.

Example Animation

The following shows an example of my programing running with a Gosper Glider Gun: **link**

(https://usu.instructure.com/courses/639065/files/79619856/download?download_frd=1)

Unit Tests

There are unit tests associated with this assignment. The provided project includes them, but they are commented out so that the sample code will compile and run. As you work on your code, you'll want to uncomment the unit tests to ensure you are passing them and implementing the logic correctly.

Notes & Submission

Use the following filenames for your code. If you have additional code files due to other patterns, that is fine, but please following the same naming convention for the files and class names (consistency is king!).

- Start with the following code framework for your project: **link** (https://usu.instructure.com/courses/639065/files/79619926/download?download_frd=1)
- Submit all of these files (and any others you create) in a single zip file. Go inside the /src folder and make a .zip file of everything in there, but ***not*** including the /com folder.
  - **ConwaysLife.java**
  - **LifeSimulator.java**
  - Pattern Files
    - **Pattern.java**
    - **PatternAcorn.java**
    - **PatternBlinker.java**

    - **PatternBlock.java**
    - **PatternGlider.java**
- Your code must compile without any warnings or compiler errors. See syllabus regarding code that has compiler errors.
- Your code must adhere to the CS 1410 coding standard: **link**
- Java SDK Docs: **link** **(https://docs.oracle.com/en/java/javase/15/)**

Assignment 7 - Rubric

Criteria	Ratings		Pts
Compiles without any warnings	**2 pts** **Full Marks**	**0 pts** **No Marks**	2 pts
Follows required course code style	**2 pts** **Full Marks**	**0 pts** **No Marks**	2 pts
Passes all unit tests	**2 pts** **Full Marks**	**0 pts** **No Marks**	2 pts
Correctly implemented render method	**5 pts** **Full Marks**	**0 pts** **No Marks**	5 pts
Correctly implemented LifeSimulator class.	**10 pts** **Full Marks**	**0 pts** **No Marks**	10 pts
PatternBlock	**2 pts** **Full Marks**	**0 pts** **No Marks**	2 pts
PatternBlinker	**2 pts** **Full Marks**	**0 pts** **No Marks**	2 pts
PatternGlider	**2 pts** **Full Marks**	**0 pts** **No Marks**	2 pts
PatternAcorn	**3 pts** **Full Marks**	**0 pts** **No Marks**	3 pts
Animation demonstration	**10 pts** **Full Marks**	**0 pts** **No Marks**	10 pts
			Total Points: 40