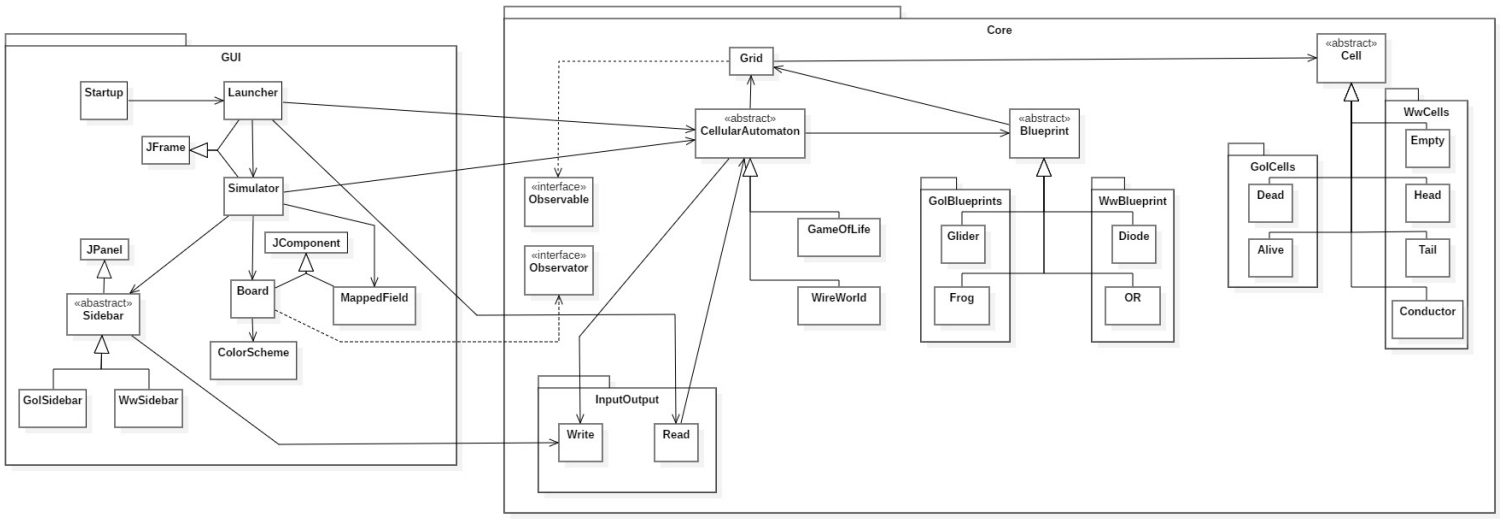
# Specyfikacja implementacyjna automatu komórkowego: `Wire World`

Sebastian Jurga, Piotr Owczarczyk

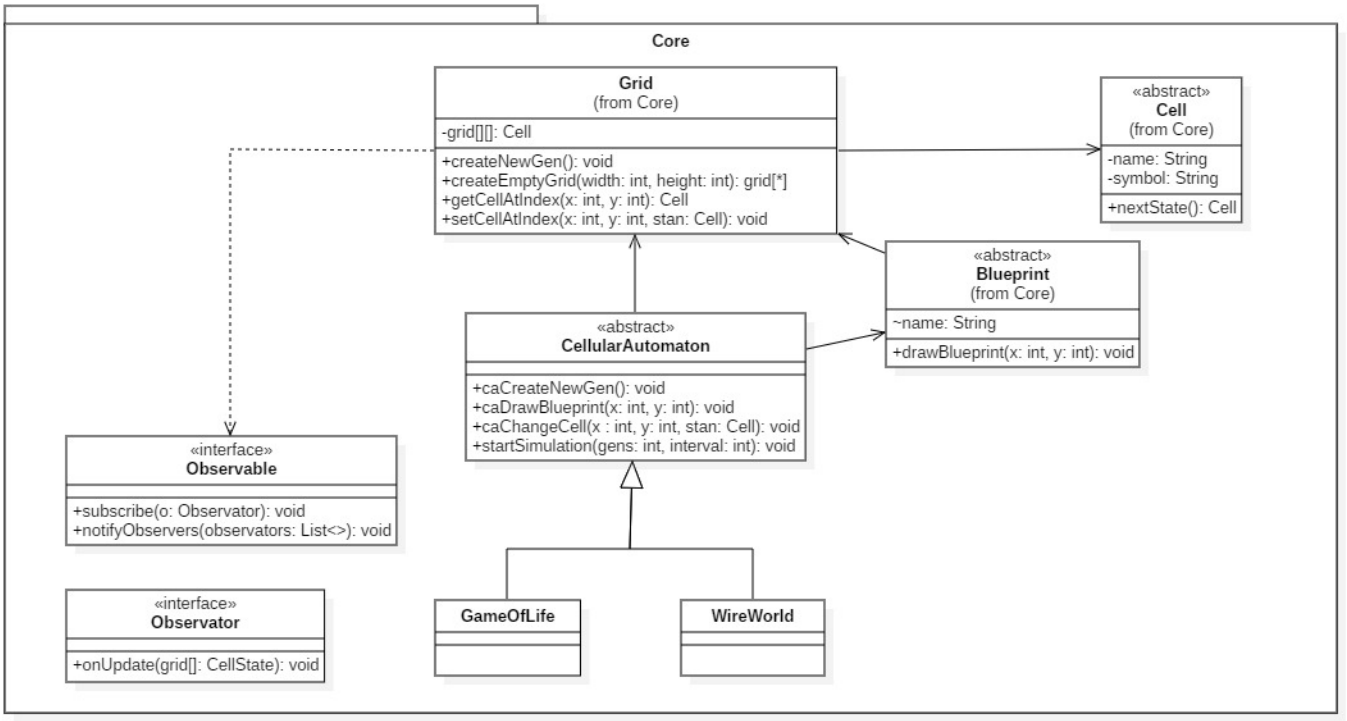
21.05.2019

## 1 Diagram modułów



# 2 Opis poszczególnych modułów

Program dzieli się na trzy pakiety, z których każdy zawiera następujące klasy:




1. Core:

   1.1. `CellularAutomaton` – Klasa odpowiadająca za sterowanie częścią logiczną programu.Referencja do tej klasy znajduje się w klasie `Simulator` pakietu `GUI` Posiada funkcje:

   a) `caCreateNewGen` – wywołuje funkcję createNewGen klasy Grid,

   b) `caDrawBlueprint` – wywołuje funkcję drawBlueprin klasy Blueprint,

   c) `caChangeCell` – wywołuje funkcję setCellAtIndex klasy Grid,

   d) `startSimulation` – uruchamia symulację automatu komórkowego z podaną liczbą generacji oraz podanym interwałem tworzenia następnych generacji.

   Klasy dziedziczące po `CellularAutomaton` to:

   a) `GameOfLife`

   b) `WireWorld`

Klasy te służą do rozróżniania, z których schematów i komórek będzie korzystał program, a także jakie będzie wyglądało menu boczne w interfejsie użytkownika.

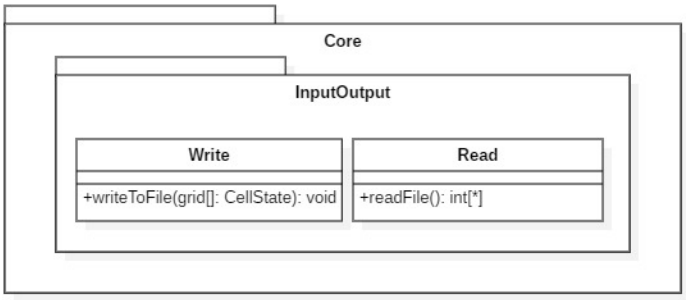
1.2. `Grid` – klasa przechowująca tablicę `grid` o elementach klasy `Cell`, w której zapisane są informacje o siatce automatu. Zawiera ona również metody służące do obsługi planszy:

- createNewGrid – tworzy kolejną generację automatu,
- createEmptyGrid – tworzy pustą siatkę o podanych wymiarach (wymiary pobierane są z pierwszego okienka interfejsu użytkownika),
- getCellAtIndex – zwraca element tablicy typu Cell o podanych współrzędnych `x` i `y`.
- setCellAtIndex – ustawia element tablicy typu Cell o podanych współrzędnych `x` i `y`, na obiekt typu Cell podanego w argumencie funkcji.

1.3. `Cell` – Klasa przechowująca informację na temat komórek. Pole `name` określa nazwę wybranej komórki, natomiast `symbol` jest znakiem rozpoznawalnym komórki. Metoda `nextState` określa stan komórki w następnej generacji. Każda klasa dziedziczące po abstrakcyjnej klasie `Cell` ma inną implementację metody nextState, ponieważ każda komórka w kolejnej generacji ma inny stan.

1.4. `Blueprint` – Obiekt przechowujący następujące informacje, na swój temat:

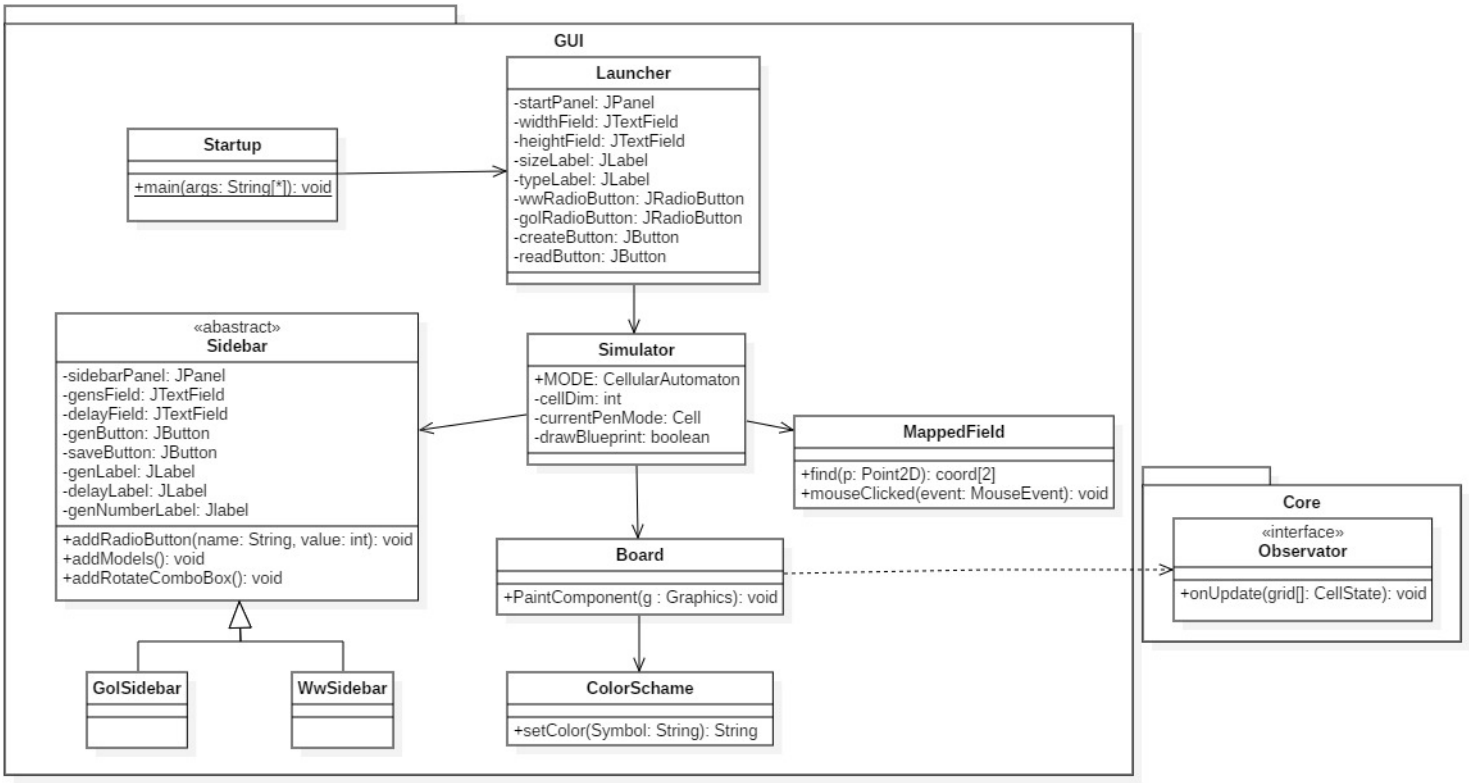
- Nazwa,
- Funkcja, umożliwiająca elementowi `Blueprint` "wklejenie się"do siatki `grid[][]`




2. InputOutput:

2.1. `Write` – Klasa zapisująca aktualną generację do pliku w postaci ciągu cyfr (dla automatu WireWorld komórki: EMPTY – 0, HEAD – 1, TAIL – 2, CONDUCTOR –3 ; dla Game of Life: DEAD – 0, ALIVE 1),

2.2. `Read` – Odczytująca zapisaną w pliku generację. Przekazuje ona przeczytane wartości do tablicy `grid`.




3. GUI:

   3.1. `Startup` – Klasa uruchamiająca program i otwierająca pierwsze okno interfejsu użytkownika.

   3.2. `Launcher` – Klasa rysująca okno startowe programu, w którym użytkownik ma możliwość wybrania wielkości siatki oraz trybu automatu, a także wyboru pliku do wczytania.

   3.3. `Simulator` – Klasa pośrednicząca pomiędzy interfejsem użytkownika a częścią logiczną programu. `Simulator` przekazuje wszystkie akcje, związane z wyborem elementu listy lub kliknięcia przycisku poprzez, referencję do obiektu `CellularAutomaton`. Okno klasy `Simulator` zawiera komponenty `Board` , `Sidebar` i `MappedField`.

   3.4. `Board` – Komponent zawierający narysowaną siatkę komórek w postaci kolorowych kwadratów. Implementuje interfejs `Observator` dzięki czemu może śledzić zmiany zachodzące w `CellularAutomaton` i na bieżąco aktualizować kolory kwadratów

3.5. `ColorSchame` – Klasa wiążąca symbol komórki z jej odpowiednikiem w postaci koloru na planszy.

3.6. `MappedField` – Komponent śledzący akcje związane z myszką. Za pomocą funkcji `find()` obliczane jest, w którym kwadracie znajduje się myszka. Informacja ta przekazywana jest do `CellularAutomaton`.

3.7. `Sidebar` – Klasa przechowująca rozmieszczenie bocznego menu. Elementy takie jak wybór liczby generacji, wybór interwału, aktualna generacja oraz przyciski *"Generuj!"* i *"Zapisz"* są wspólne dla każdego trybu automatu. Natomiast pozostałe elementy jak wybór trybu `Pisaka`, lista gotowych `Modeli` i menu wyboru obrotu `Modeli` będą tworzone za pomocą funkcji dostępnych w klasie `SideBar` odpowiednio w podklasach:

a) `GolSidebar`

b) `WwSidebar`

# 3 Opis przepływu sterowania

1. `Startup` – Uruchomienie programu, wyświetlenie okna `Launcher`;

2. `Launcher:`

   a) Pobranie od użytkownika wymiarów siatki do utworzenia i przekazanie ich do funkcji `createEmptyGrid()`, klasy `Grid`, lub

   b) Otwarcie okna wyboru plików, a następnie przekazanie pliku do `Read`, który następnie:

      I. Odczyta z pliku wymiary siatki,

      II. Wywoła funkcję `createEmptyGrid()`, klasy `Grid`,

      III. Wypełni przy pomocy `Grid.setCellAtIndex()` całą planszę.

   Następnie otworzy okno `Simulator` ;

3. `Simulator:`

   3.1. Dodanie do okna komponentów:

   - `Sidebar`
   - `Board`
   - `MappedField`

   3.2. `MappedField` – Obsługa edycji aktualnej generacji:

   a) W przypadku kliknięcia kursorem, obliczenie ze współrzędnych kliknięcia, współrzędnych komórki w tablicy, przekazanie jej do `Simulator`, który po sprawdzeniu zmiennej `drawBlueprint` (zmienianej na `true` w przypadku wybrania gotowego modelu z menu i zmienianej z powrotem na `false` po jego wklejeniu):

- (w przypadku drawBlueprint == false) – prześle je wraz z `currentPenMode` do klasy `CellularAutomaton`, która wywoła funkcję `Grid.setCellIndex()` z odpowiednimi wartościami, bądź
- (gdy drawBlueprint == true) – sprawdzi aktualnie wybrany `Blueprint`, i prześle jego nazwę wraz ze współrzędnymi do `CellularAutomaton`, który wywoła funkcję `drawBlueprint()` w odpowiedniej klasie `Blueprint`, używającej funkcji `setCellAtIndex()` w klasie `Grid`, odpowiednio się "wklei"

b) `Board` – Po otrzymaniu wiadomości z funkcji `notifyObservers()` w przypadku gdy zostanie zmieniony stan komórki przez `Grid.setCellAtIndex()`, bądź po wygenerowaniu całej nowej generacji – pobieranie danych z `Grid` na temat aktualnej generacji oraz przetwarzanie jej na obraz graficzny.

c) `Sidebar` – Przesłanie do `CellularAutomaton` informacji dotyczących:
- Ilości generacji do przeprowadzenia,
- Opóźnienia między wyświetlaniem kolejnych iteracji,
- Sygnału do rozpoczęcia symulacji.

4. `CellularAutomaton` – Rozpoczęcie symulacji:

4.1. Wywołanie odpowiednią ilość razy funkcji `Grid.createNewGen()`, która:

4.1.1. Skopiuje aktualną generację do tablicy tymczasowej `temp`,

4.1.2. Dla każdej komórki wywoła funkcję `nextState()` i zapisze zwracaną przez nią wartość do `grid[][]`

4.2. Poinformowanie `Obserwatorów` po wygenerowaniu każdej kolejnej iteracji, z odpowiednim opóźnieniem.

5. `Board` – Aktualizacja wyświetlanej generacji,

6. `Sidebar` – Wywołanie funkcji `writeToFile()`;

7. `Write:`

7.1. Otwarcie okna wyboru plików,

7.2. Pobranie aktualnej generacji z `CellularAutomaton`,

7.3. Zapisanie aktualnej generacji do wybranego przez użytkownika pliku.

# 4 Testy modułów

Podczas tworzenia programu przeprowadzone zostaną testy następujących metod:

1. `writeToFile`: do przeprowadzenia testu niezbędne będzie stworzenie tablicy `grid`. Test będzie miał za zadanie sprawdzić, czy dane zapisane w pliku o rozszerzeniu .txt są takie same jak wartości przekazane do tablicy. W zależności od trybu `DEAD` i `ALIVE` są reprezentowane przez `0` i `1` oraz `EMPTY`, `HEAD`, `TAIL`, `CONDUCTOR` są odpowiednio `0`, `1`, `2` i `3`,

2. `readFile`: funkcji przekazany zostanie plik w następującej formie:

   ```
   WW
   5 10
   0030000000
   0030000000
   0030000000
   0010000000
   0020000000
   DIODE: 0, 2, H NORMAL
   ```

   Poprawna wartość zwracana:

   ```
   0030000000
   0033300000
   3333033300
   0013300000
   0020000000
   ```

3. `drawBlueprint` klasy `Glider`. Podczas testowania musi zostać utworzona pusta tablica `grid` na której zostanie dodany gotowy schemat. Funkcji należy przekazać wartości `x` i `y`, które oznaczają początkową komórkę od której rozpoczyna się rysowanie schematu. Pusta tablica `5x5` :

   ```
   00000
   00000
   00000
   00000
   00000
   ```

   Tablica po wklejeniu schematu w komórkę `(1,1)`:

   ```
   00000
   00100
   00010
   01110
   00000
   ```

4. `nextState` klasy `Conductor`. Funkcja, w tej klasie, oblicza stan komórki w kolejnej generacji na podstawie komórek sąsiadujących, dlatego potrzebna jest wcześniej stworzona tablica `grid`. Przykładowa tablica:

```
00200
00100
21333
00300
00300
```

W tym teście interesuje nas komórka środkowa czyli `(2,2)`. Komórka sąsiaduje z dwiema głowami elektronu dlatego oczekiwany rezultat to komórka `HEAD`.