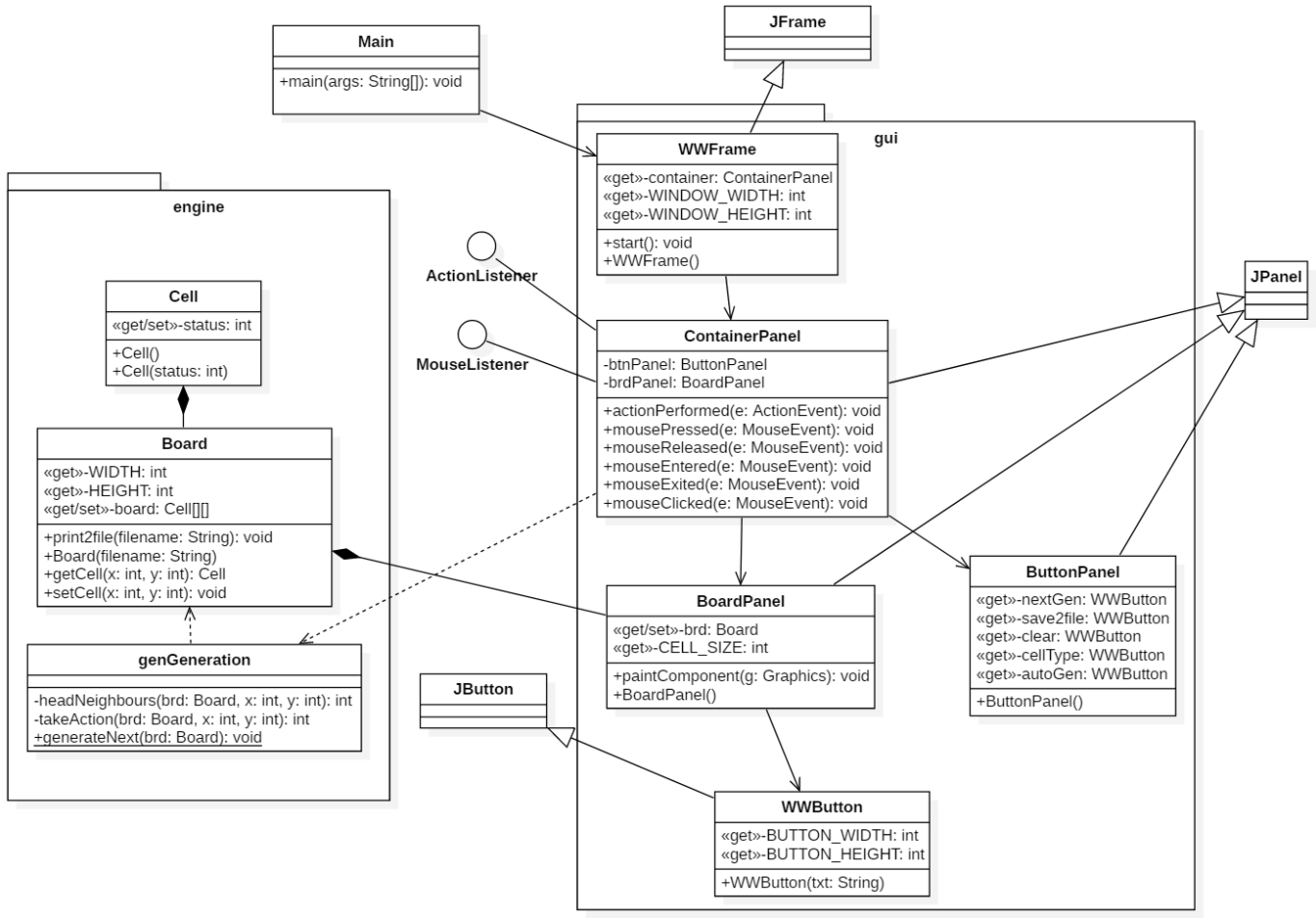
Raport końcowy programu *WireWorld*

Krzysztof Maciejewski
Hubert Kunikowski

9 czerwca 2019

Spis treści

1 Ostateczny diagram klas



Ostateczna struktura programu pozostała niemal bez zmian w stosunku do wersji ze specyfikacji implementacyjnej.

- *Cell* - przechowuje status komórki.
- *Board* - przechowuje tablicę komórek (obiektów *Cell*).
- *genGenerator* - odpowiada za generowanie kolejnych generacji.
- *WWFrame* - dziedziczy po *JFrame*, odpowiada za wyświetlanie okna.
- *ContainerPanel* - dziedziczy po JPanel, przechowuje zawartość okna.
- *ButtonPanel* - dziedziczy po JPanel, odpowiada za wyświetlanie i działanie przycisków.
- *BoardPanel* - dziedziczy po JPanel, odpowiada za wyświetlanie i działanie planszy.
- *WWButton* - dziedziczy po JButton, określa właściwości przycisków.

2 Opis modyfikacji

1. **Klasa ButtonPanel**

   Klasa ta początkowo miała posiadać przycisk "Auto generate". Przycisk ten miał automatycznie generować kolejne generacje. Ostatecznie nie został on dodany, gdyż nie starczyło nam czasu.Klasa ta zawiera dodatkowo dwa atrybuty *Jlabel*, które odpowiadają za wyświetlenie tytułu (*WireWorld*) oraz autorów w oknie.

2. **Pakiety**

   Ostatecznie pakiety *engine* oraz *gui* nie zostały utworzone, gdyż po rozdzieleniu klas do pakietów napotkaliśmy na komplikacje, którym nie byliśmy w stanie zaradzić.

3 Działanie

Program możemy wywołać bez argumentów lub z argumentem będącym nazwą pliku. Jeżeli podamy nazwę pliku, program wczyta z niego początkowy układ komórek. Domyślnie program wczyta gotową planszę z folderu *resources*.

Z boku mamy cztery dostępne przyciski.

- Przycisk "Next Generation" następną nową generację.
- Przycisk "Save to file" zapisuje bieżącą generację do pliku tekstowego.
- Przycisk "Clear" czyści planszę.
- Ostatni przycisk może znajdować się w jednym z trzech stanów: *Connector*, *Electron Head* oraz *Electron Tail*. Stan tego przycisku określa jaką komórkę umieszczamy na planszy.

Aby zmienić stan komórki, klikamy na wybraną komórkę na planszy.

4 Podsumowanie testów

W celu przetestowania stworzonych klas i metod, napisaliśmy testy jednostkowe korzystając z biblioteki *JUnit*. Poprawne wykonanie sprawdzane było poprzez porównanie danych otrzymanych z oczekiwanymi dla różnych metod. Pojedyncze wykonanie testu bez komunikatu o błędzie świadczy o jego poprawnym wykonaniu. Powstały testy klas *GenGenerator*, *Cell*, *Board*. Każda metoda testująca wykorzystywała stworzenie nowego obiektu danej klasy i wykonywanie na nim zadanych metod i sprawdzanie poprawności wyników. Poprawność wyników sprawdzana dzięki asercjom dostępnym w bibliotece *JUnit*, które są metodami statycznymi w klasie *Assert*. Najczęściej stosowana była asercja *assertEquals*, która przyjmuje dwa parametry - wartość oczekiwaną i rzeczywistą, a następnie porównuje je ze sobą. Jeśli wartości się różnią, rzuca wyjątek. Przykładowy test:

```
import org.junit.Assert;
import org.junit.Test;

import static org.junit.Assert.*;

public class genGeneratorTest extends genGenerator {

    @Test
    public void TestgenerateNext() {
    Board TestBoard;
    TestBoard = new Board( file_name: "resources/" + "TestBoard.txt");
    Cell Cell_4 = new Cell( status: 4);
    Cell Cell_1 = new Cell( status: 1);
    Cell Cell_2 = new Cell( status: 2);
    Cell Cell_3 = new Cell( status: 3);
    TestBoard.getBoard()[1][1] = Cell_1;
    TestBoard.getBoard()[3][3] = Cell_2;
    TestBoard.getBoard()[5][5] = Cell_3;
    TestBoard.getBoard()[7][7] = Cell_4;
    generateNext(TestBoard);
    Assert.assertEquals( expected: 1, TestBoard.getCell( x: 1, y: 1).getStatus());
    Assert.assertEquals( expected: 3, TestBoard.getCell( x: 3, y: 3).getStatus());
    Assert.assertEquals( expected: 4, TestBoard.getCell( x: 5, y: 5).getStatus());
    Assert.assertEquals( expected: 4, TestBoard.getCell( x: 7, y: 7).getStatus());
    Assert.assertEquals( expected: 0, TestBoard.getCell( x: 10, y: 10).getStatus());
    }
}
```