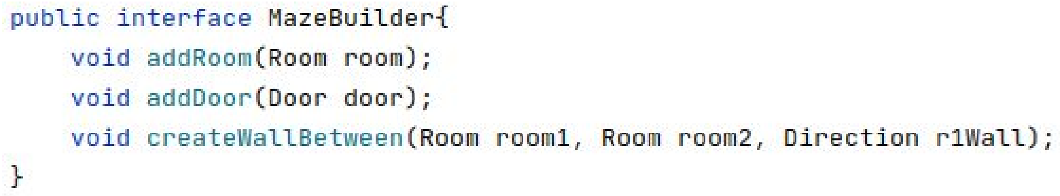
Sprawozdanie z ćwiczenia 3

Krzysztof Nalepa, Jakub Solecki

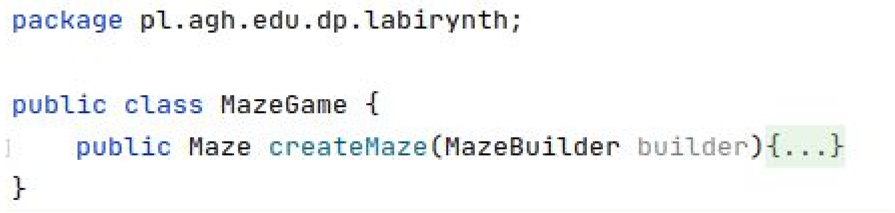
Builder

1. Interfejs MazeBuilder



```
public interface MazeBuilder{
    void addRoom(Room room);
    void addDoor(Door door);
    void createWallBetween(Room room1, Room room2, Direction r1Wall);
}
```

2. Modyfikacja funkcji składowej



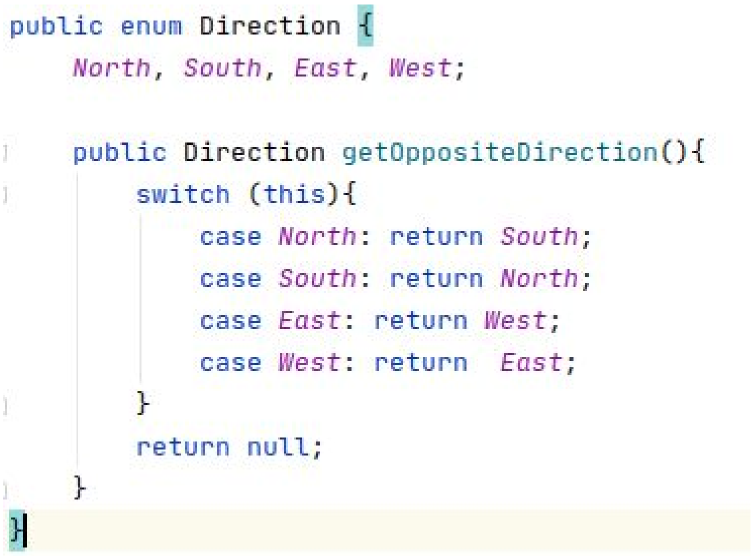
```
package pl.agh.edu.dp.labirynth;

public class MazeGame {
    public Maze createMaze(MazeBuilder builder){...}
}
```

3. Interpretacja

Dzięki temu możemy budować bardziej skomplikowany obiekt w oddzielnej klasie. Sam proces konstrukcyjny przebiega sprawniej i jest bardziej czytelny. Ponadto unikamy powtarzania tego samego kody

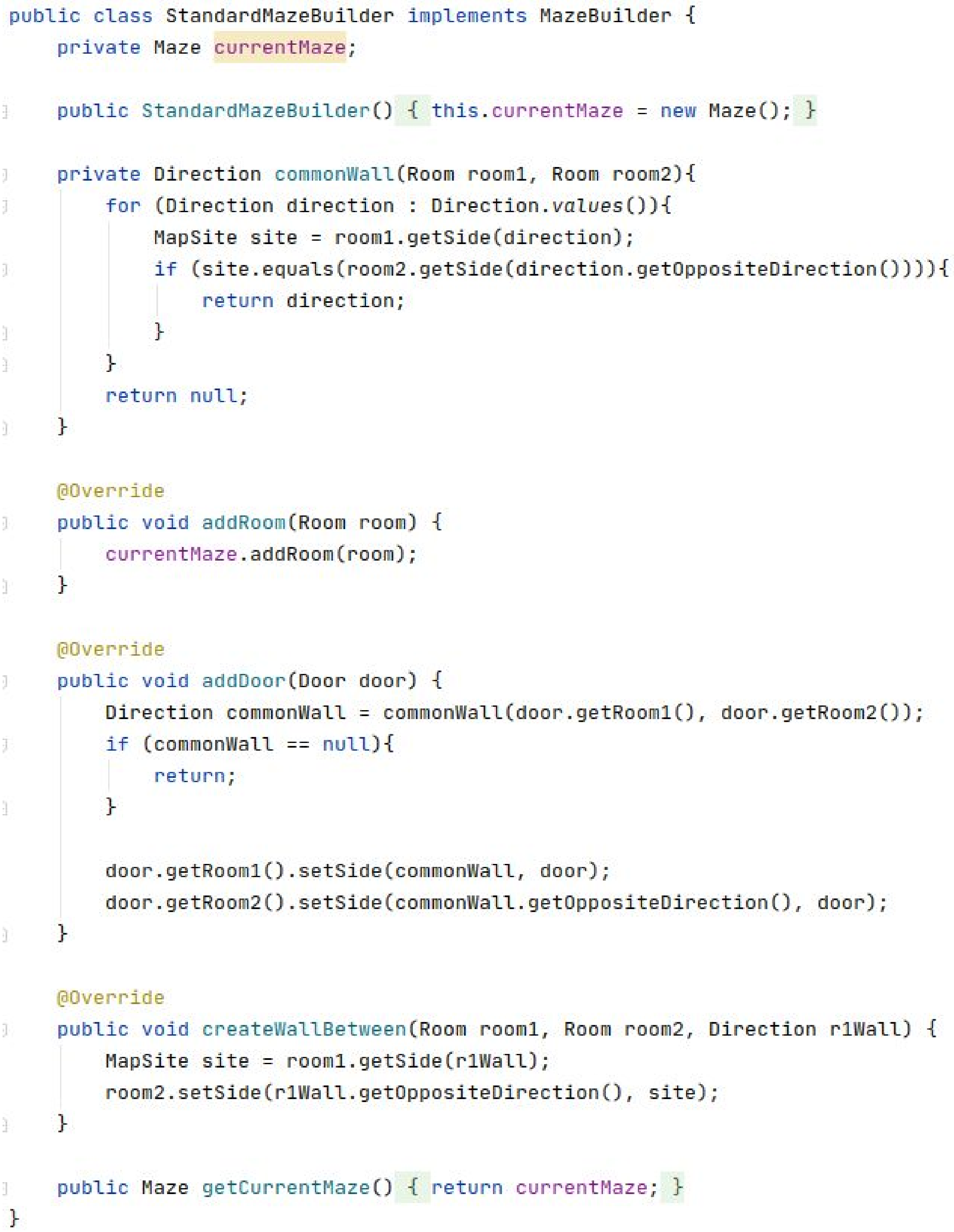
4. StandardBuilderMaze

Dodatkowo dodaliśmy metodę getOppositeDirection w enum Direction



```java
public enum Direction {
    North, South, East, West;

    public Direction getOppositeDirection(){
        switch (this){
            case North: return South;
            case South: return North;
            case East: return West;
            case West: return  East;
        }
        return null;
    }
}
```

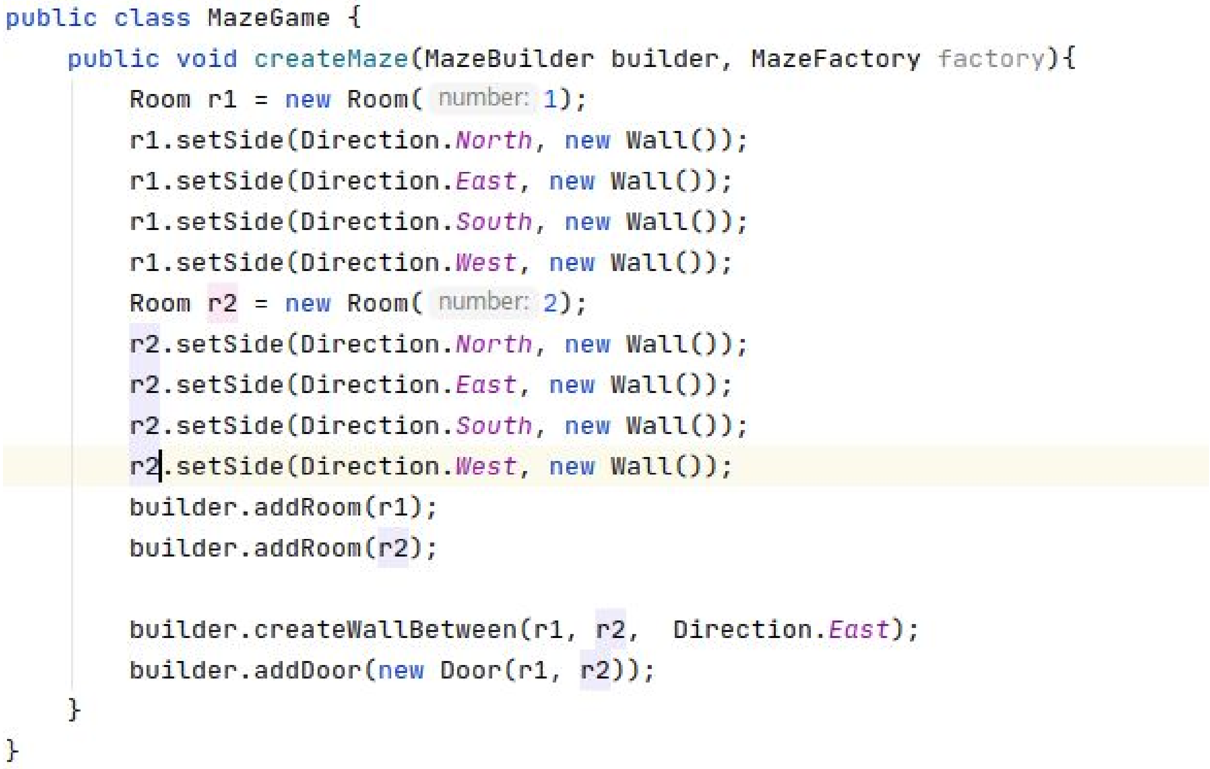
Następnie dodaliśmy klasę StandardMazeBuilder



```
public class StandardMazeBuilder implements MazeBuilder {
    private Maze currentMaze;

    public StandardMazeBuilder() { this.currentMaze = new Maze(); }

    private Direction commonWall(Room room1, Room room2){
        for (Direction direction : Direction.values()){
            MapSite site = room1.getSide(direction);
            if (site.equals(room2.getSide(direction.getOppositeDirection()))){
                return direction;
            }
        }
        return null;
    }

    @Override
    public void addRoom(Room room) {
        currentMaze.addRoom(room);
    }

    @Override
    public void addDoor(Door door) {
        Direction commonWall = commonWall(door.getRoom1(), door.getRoom2());
        if (commonWall == null){
            return;
        }

        door.getRoom1().setSide(commonWall, door);
        door.getRoom2().setSide(commonWall.getOppositeDirection(), door);
    }

    @Override
    public void createWallBetween(Room room1, Room room2, Direction r1Wall) {
        MapSite site = room1.getSide(r1Wall);
        room2.setSide(r1Wall.getOppositeDirection(), site);
    }

    public Maze getCurrentMaze() { return currentMaze; }
}
```

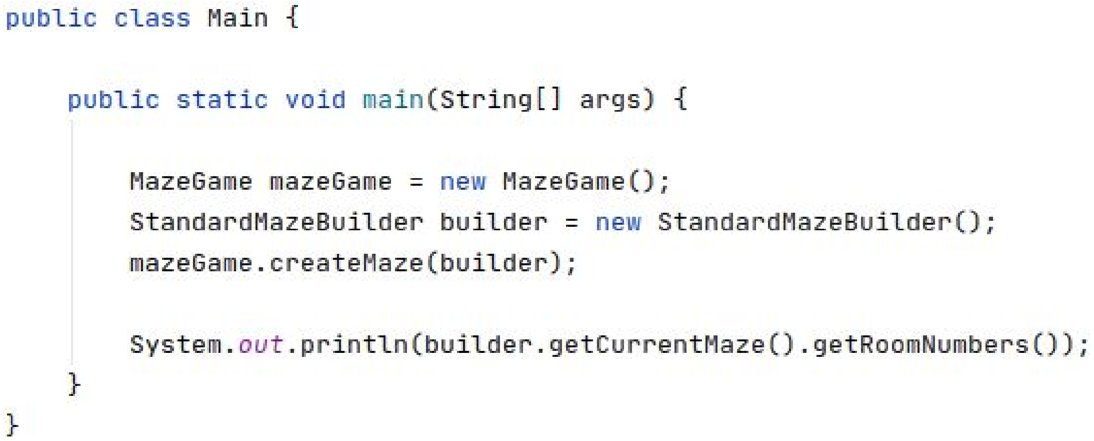
5. Tworzenie Labiryntu

Zmodyfikowaliśmy metodę createMaze w klasie MazeGame



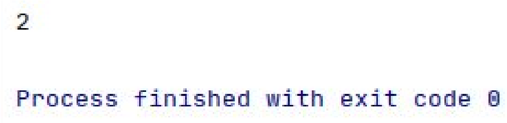
```java
public class MazeGame {
    public void createMaze(MazeBuilder builder, MazeFactory factory){
        Room r1 = new Room( number: 1);
        r1.setSide(Direction.North, new Wall());
        r1.setSide(Direction.East, new Wall());
        r1.setSide(Direction.South, new Wall());
        r1.setSide(Direction.West, new Wall());
        Room r2 = new Room( number: 2);
        r2.setSide(Direction.North, new Wall());
        r2.setSide(Direction.East, new Wall());
        r2.setSide(Direction.South, new Wall());
        r2.setSide(Direction.West, new Wall());
        builder.addRoom(r1);
        builder.addRoom(r2);

        builder.createWallBetween(r1, r2,  Direction.East);
        builder.addDoor(new Door(r1, r2));
    }
}
```

Następnie zmodyfikowaliśmy maina i uruchomiliśmy program



```java
public class Main {

    public static void main(String[] args) {

        MazeGame mazeGame = new MazeGame();
        StandardMazeBuilder builder = new StandardMazeBuilder();
        mazeGame.createMaze(builder);

        System.out.println(builder.getCurrentMaze().getRoomNumbers());
    }
}
```

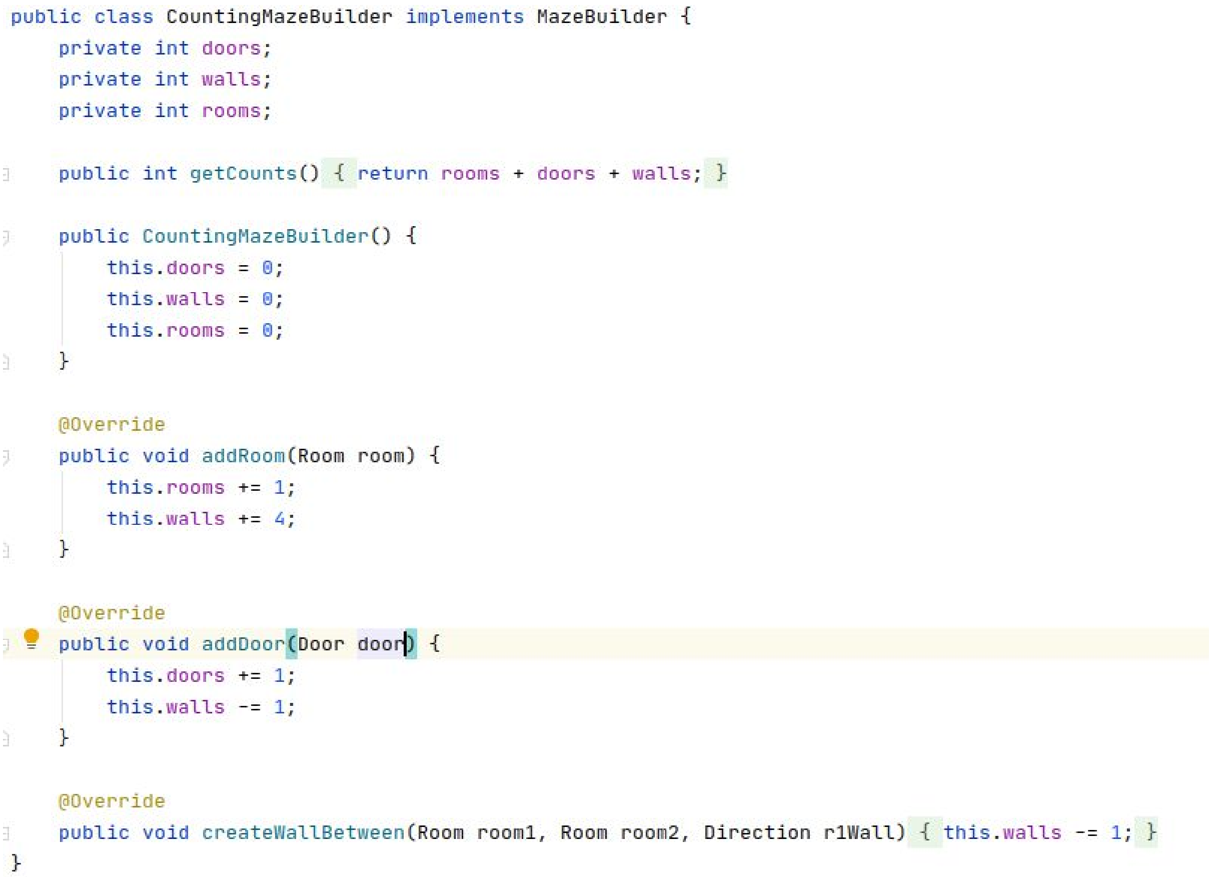
Wynik uruchomienia programu



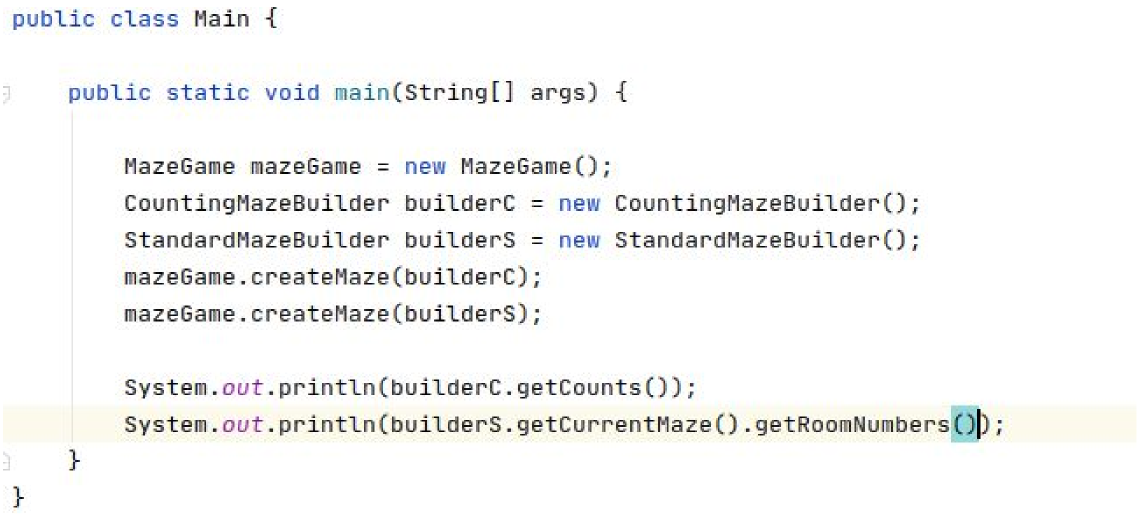
```
2

Process finished with exit code 0
```

6. CountingMazeBuilder

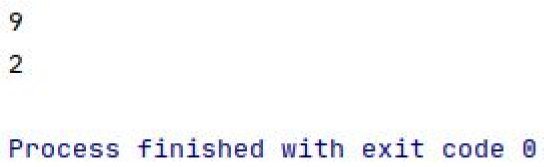
Stworzyliśmy klasę CountingMazeBuilder



```java
public class CountingMazeBuilder implements MazeBuilder {
    private int doors;
    private int walls;
    private int rooms;

    public int getCounts() { return rooms + doors + walls; }

    public CountingMazeBuilder() {
        this.doors = 0;
        this.walls = 0;
        this.rooms = 0;
    }

    @Override
    public void addRoom(Room room) {
        this.rooms += 1;
        this.walls += 4;
    }

    @Override
    public void addDoor(Door door) {
        this.doors += 1;
        this.walls -= 1;
    }

    @Override
    public void createWallBetween(Room room1, Room room2, Direction r1Wall) { this.walls -= 1; }
}
```



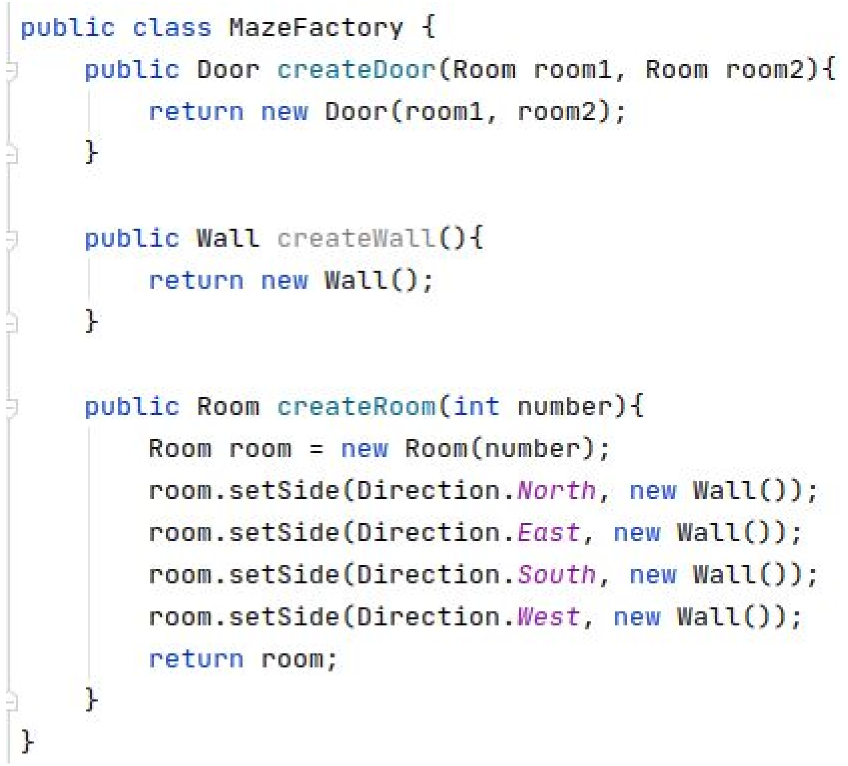
```java
public class Main {

    public static void main(String[] args) {

        MazeGame mazeGame = new MazeGame();
        CountingMazeBuilder builderC = new CountingMazeBuilder();
        StandardMazeBuilder builderS = new StandardMazeBuilder();
        mazeGame.createMaze(builderC);
        mazeGame.createMaze(builderS);

        System.out.println(builderC.getCounts());
        System.out.println(builderS.getCurrentMaze().getRoomNumbers());
    }
}
```



```
9
2

Process finished with exit code 0
```

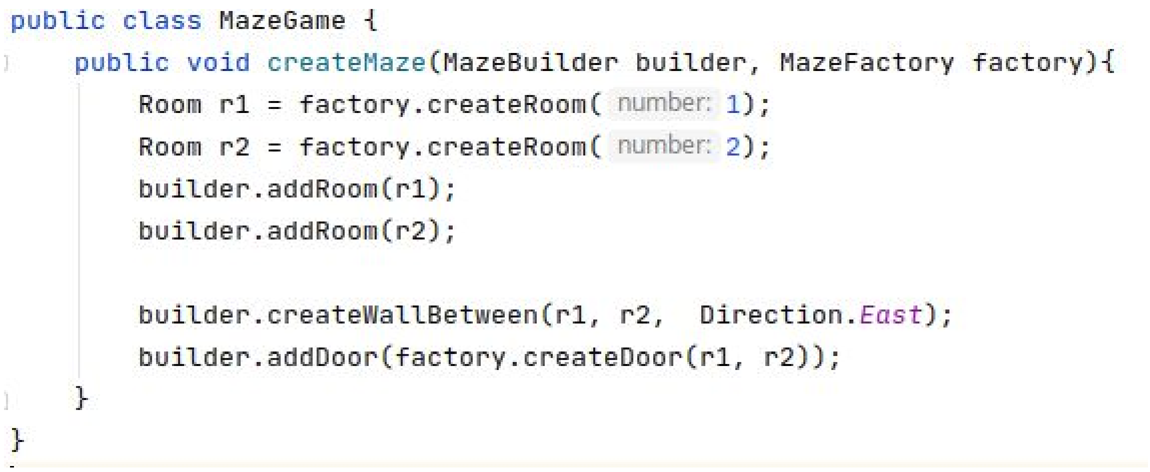
Fabryka Abstrakcyjna

1. MazeFactory



```
public class MazeFactory {
    public Door createDoor(Room room1, Room room2){
        return new Door(room1, room2);
    }

    public Wall createWall(){
        return new Wall();
    }

    public Room createRoom(int number){
        Room room = new Room(number);
        room.setSide(Direction.North, new Wall());
        room.setSide(Direction.East, new Wall());
        room.setSide(Direction.South, new Wall());
        room.setSide(Direction.West, new Wall());
        return room;
    }
}
```

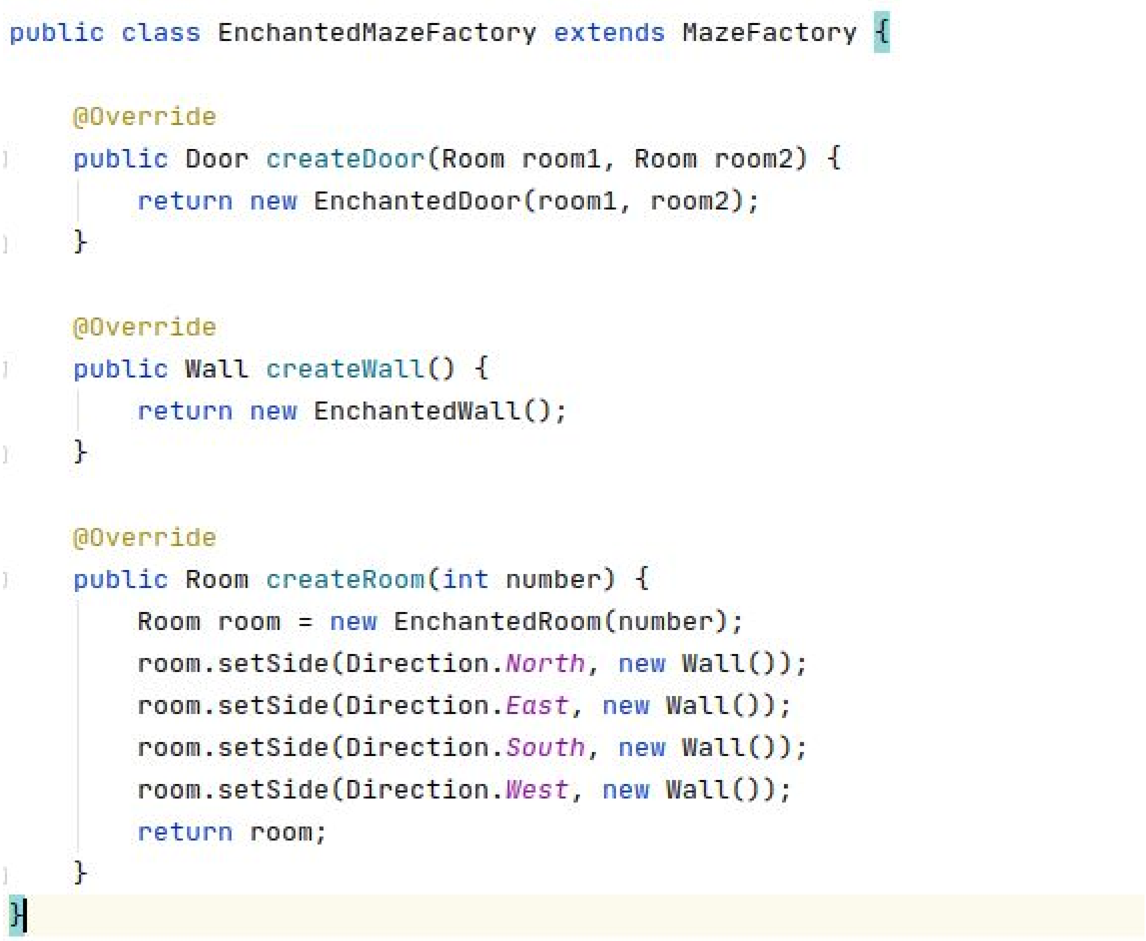
2. Modyfikacja createMaze



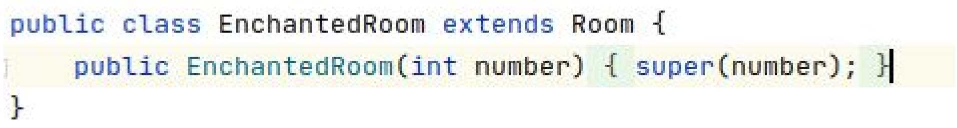
```
public class MazeGame {
    public void createMaze(MazeBuilder builder, MazeFactory factory){
        Room r1 = factory.createRoom( number: 1);
        Room r2 = factory.createRoom( number: 2);
        builder.addRoom(r1);
        builder.addRoom(r2);

        builder.createWallBetween(r1, r2,  Direction.East);
        builder.addDoor(factory.createDoor(r1, r2));
    }
}
```

3. EnchantedMazeFactory

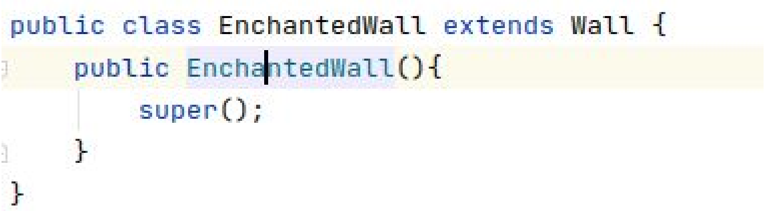
Stworzyliśmy klasę EnchantedMazeFactory oraz dla każdego z elementów labiryntu jego zaczarowany odpowiednik



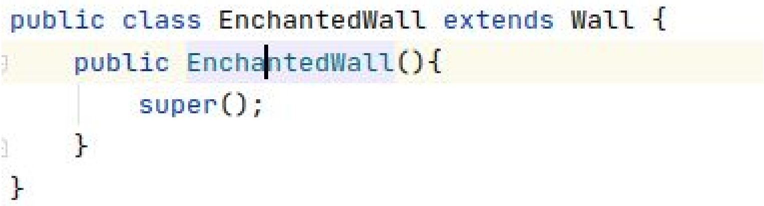
```java
public class EnchantedMazeFactory extends MazeFactory {

    @Override
    public Door createDoor(Room room1, Room room2) {
        return new EnchantedDoor(room1, room2);
    }

    @Override
    public Wall createWall() {
        return new EnchantedWall();
    }

    @Override
    public Room createRoom(int number) {
        Room room = new EnchantedRoom(number);
        room.setSide(Direction.North, new Wall());
        room.setSide(Direction.East, new Wall());
        room.setSide(Direction.South, new Wall());
        room.setSide(Direction.West, new Wall());
        return room;
    }
}
```



```java
public class EnchantedRoom extends Room {
    public EnchantedRoom(int number) { super(number); }
}
```

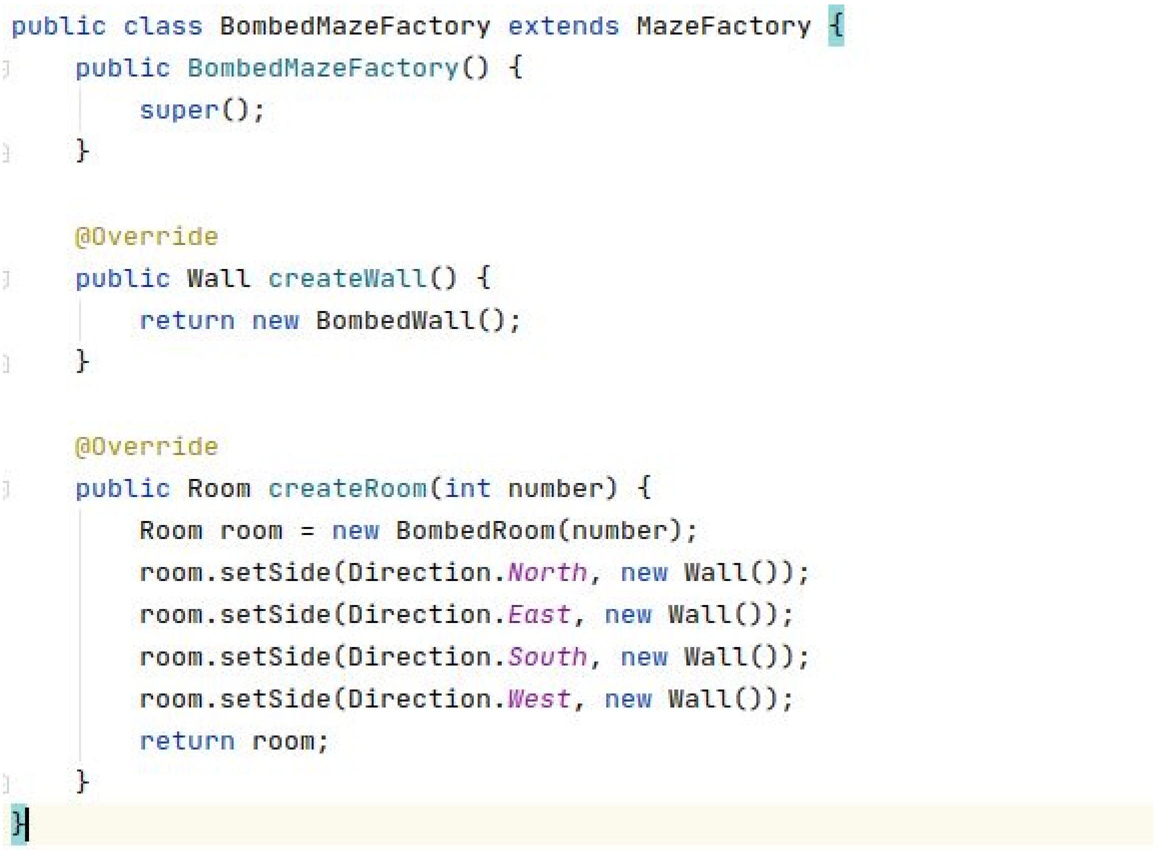


```java
public class EnchantedWall extends Wall {
    public EnchantedWall(){
        super();
    }
}
```

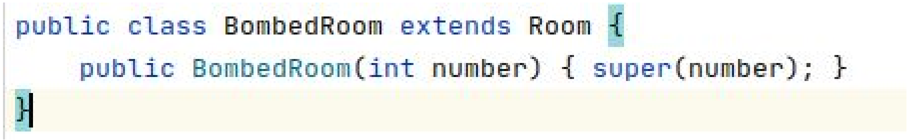


```java
public class EnchantedWall extends Wall {
    public EnchantedWall(){
        super();
    }
}
```

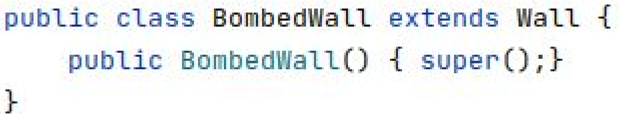
4. BombedMazeFactory



```
public class BombedMazeFactory extends MazeFactory {
    public BombedMazeFactory() {
        super();
    }

    @Override
    public Wall createWall() {
        return new BombedWall();
    }

    @Override
    public Room createRoom(int number) {
        Room room = new BombedRoom(number);
        room.setSide(Direction.North, new Wall());
        room.setSide(Direction.East, new Wall());
        room.setSide(Direction.South, new Wall());
        room.setSide(Direction.West, new Wall());
        return room;
    }
}
```



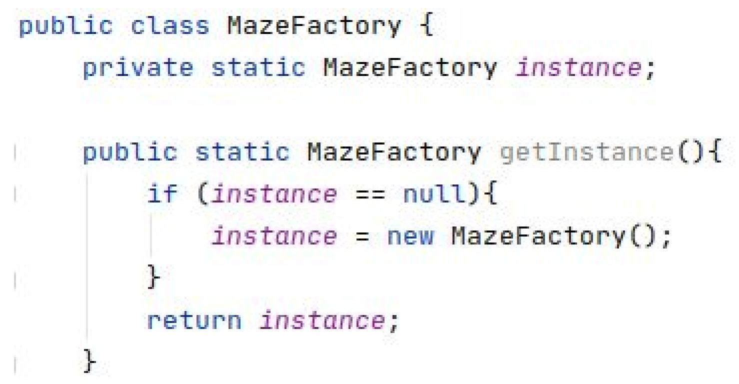
```
public class BombedRoom extends Room {
    public BombedRoom(int number) { super(number); }
}
```



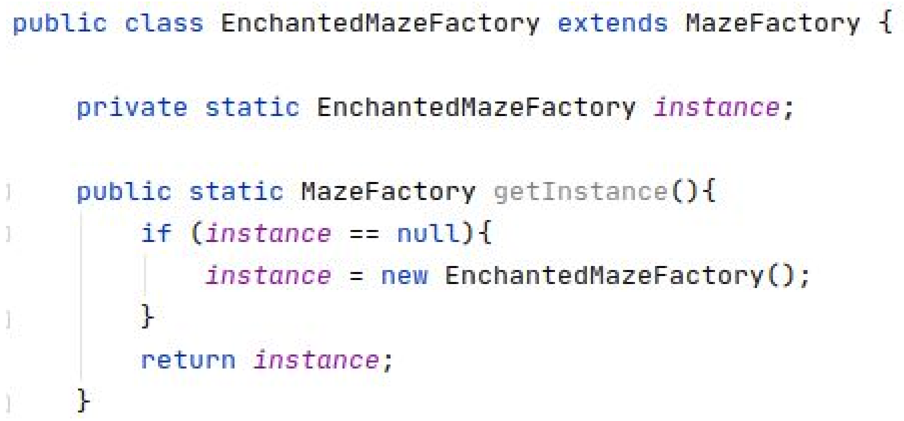
```
public class BombedWall extends Wall {
    public BombedWall() { super();}
}
```

Singleton

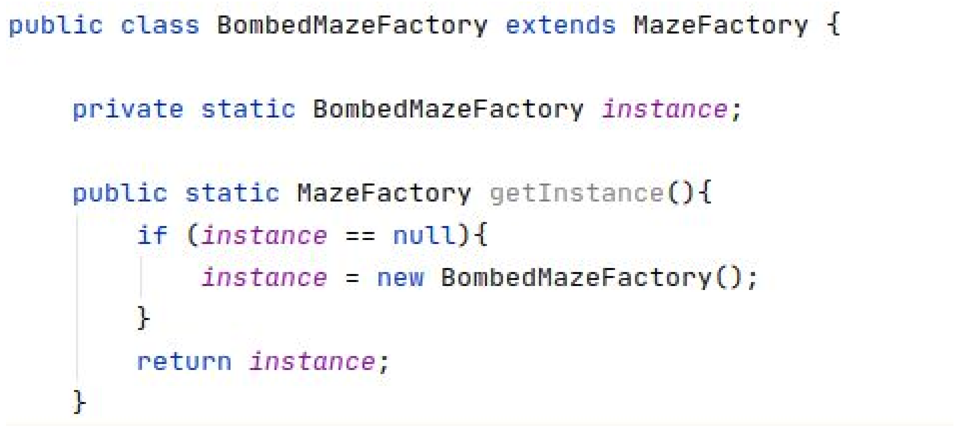
Zamieniliśmy wszystkie maze factory na singletony



```java
public class MazeFactory {
    private static MazeFactory instance;

    public static MazeFactory getInstance(){
        if (instance == null){
            instance = new MazeFactory();
        }
        return instance;
    }
```



```java
public class EnchantedMazeFactory extends MazeFactory {

    private static EnchantedMazeFactory instance;

    public static MazeFactory getInstance(){
        if (instance == null){
            instance = new EnchantedMazeFactory();
        }
        return instance;
    }
```

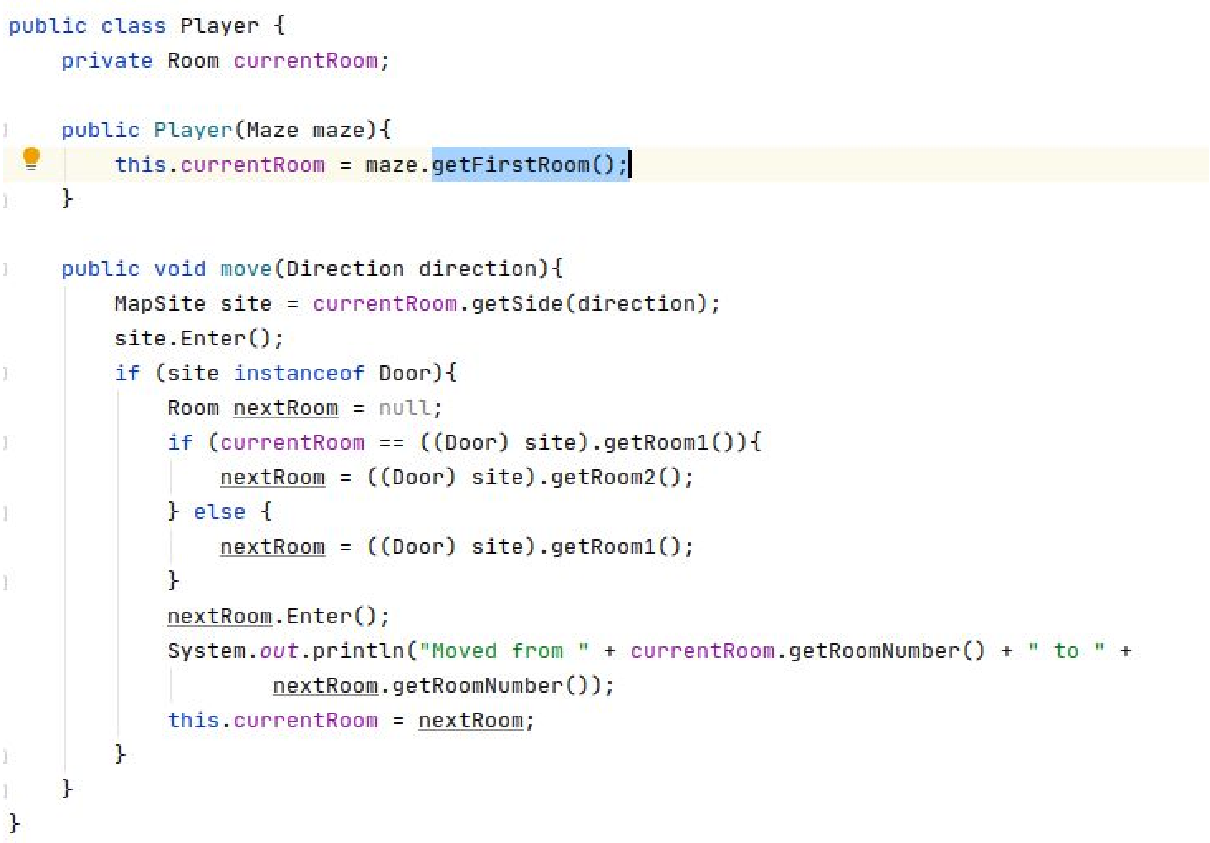


```java
public class BombedMazeFactory extends MazeFactory {

    private static BombedMazeFactory instance;

    public static MazeFactory getInstance(){
        if (instance == null){
            instance = new BombedMazeFactory();
        }
        return instance;
    }
```

Rozszerzenie aplikacji labirynt

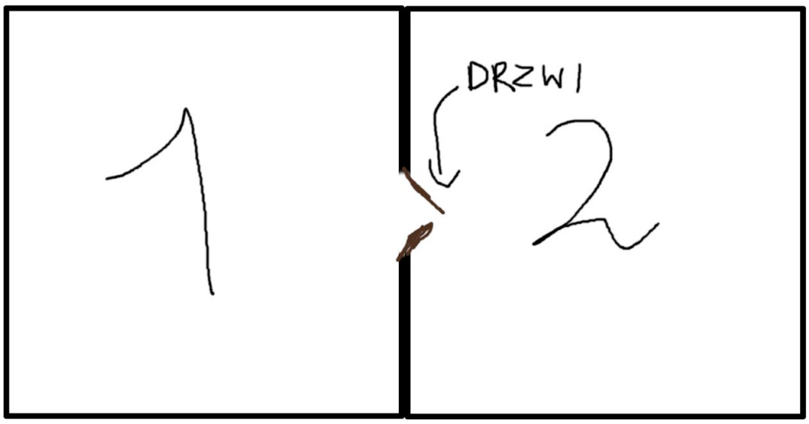
a) Przemieszczanie się po labiryncie

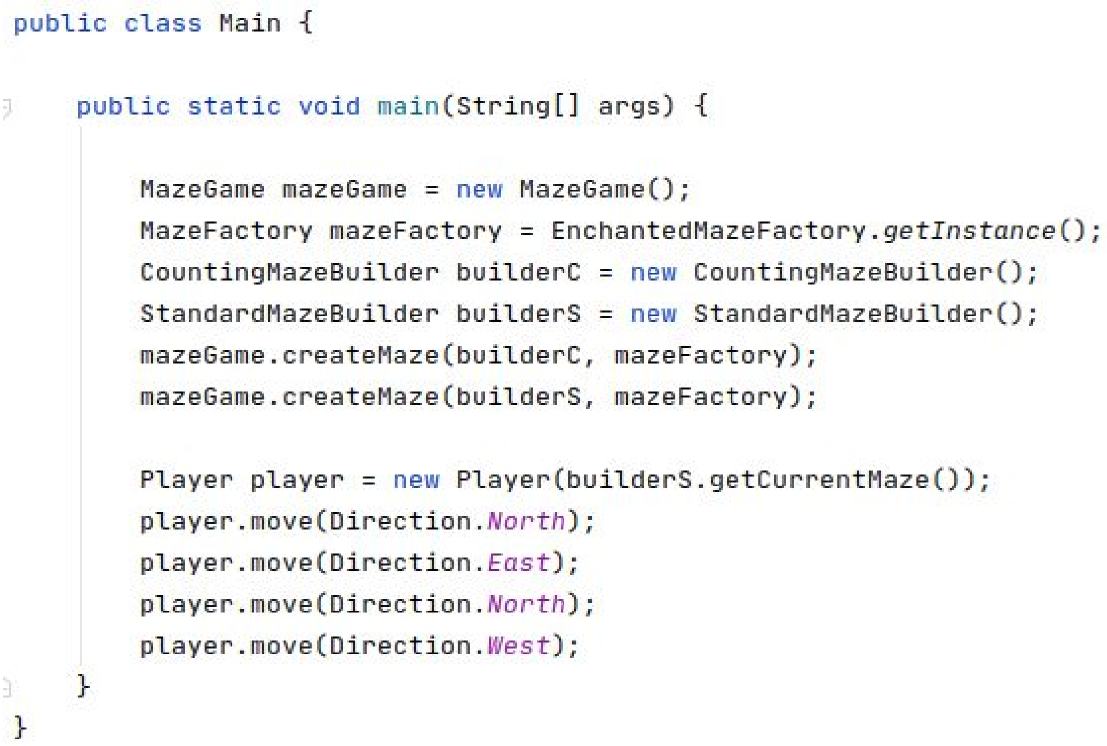
Stworzyliśmy klasę player oraz uzupełniliśmy metody Enter w każdej z klas elementów labiryntu



```java
public class Player {
    private Room currentRoom;

    public Player(Maze maze){
        this.currentRoom = maze.getFirstRoom();
    }

    public void move(Direction direction){
        MapSite site = currentRoom.getSide(direction);
        site.Enter();
        if (site instanceof Door){
            Room nextRoom = null;
            if (currentRoom == ((Door) site).getRoom1()){
                nextRoom = ((Door) site).getRoom2();
            } else {
                nextRoom = ((Door) site).getRoom1();
            }
            nextRoom.Enter();
            System.out.println("Moved from " + currentRoom.getRoomNumber() + " to " +
                    nextRoom.getRoomNumber());
            this.currentRoom = nextRoom;
        }
    }
}
```

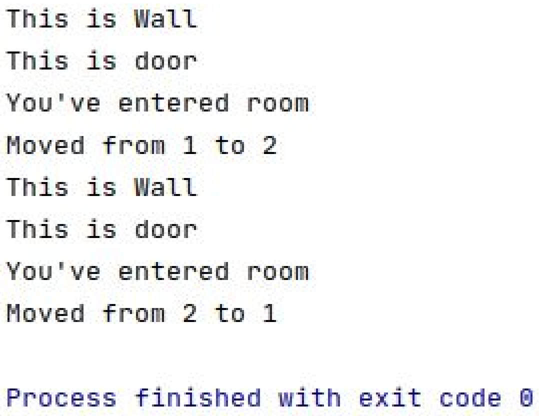
Następnie stworzyliśmy przykładowy labirynt oraz zaprogramowaliśmy ruch playera w celu testowania





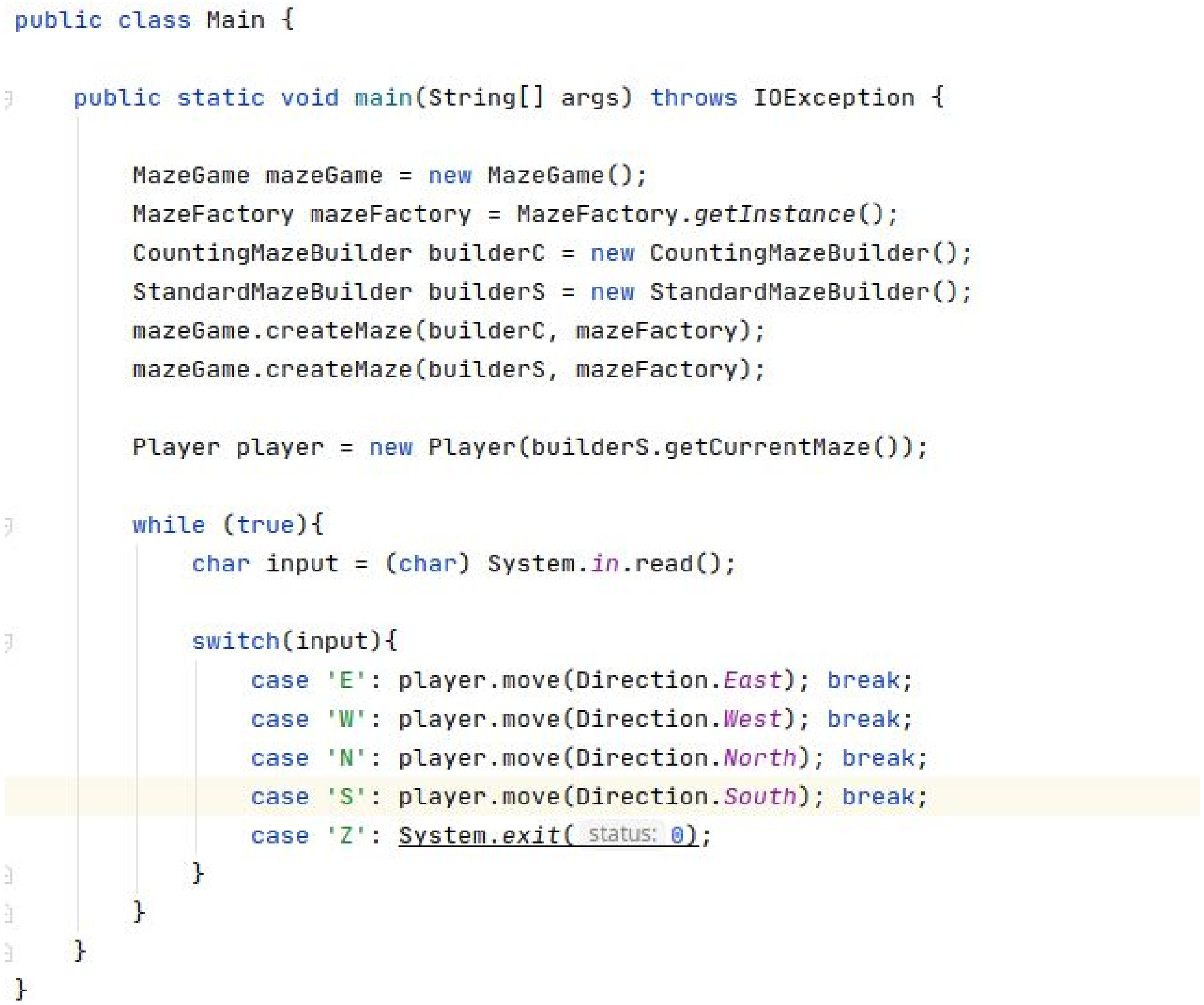
```java
public class Main {

    public static void main(String[] args) {

        MazeGame mazeGame = new MazeGame();
        MazeFactory mazeFactory = EnchantedMazeFactory.getInstance();
        CountingMazeBuilder builderC = new CountingMazeBuilder();
        StandardMazeBuilder builderS = new StandardMazeBuilder();
        mazeGame.createMaze(builderC, mazeFactory);
        mazeGame.createMaze(builderS, mazeFactory);

        Player player = new Player(builderS.getCurrentMaze());
        player.move(Direction.North);
        player.move(Direction.East);
        player.move(Direction.North);
        player.move(Direction.West);
    }
}
```

Wynik działania dla powyższego programu



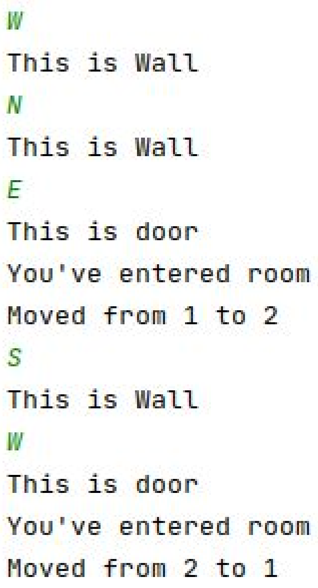
```
This is Wall
This is door
You've entered room
Moved from 1 to 2
This is Wall
This is door
You've entered room
Moved from 2 to 1

Process finished with exit code 0
```

Następnie dodaliśmy sterowanie tekstem do maina



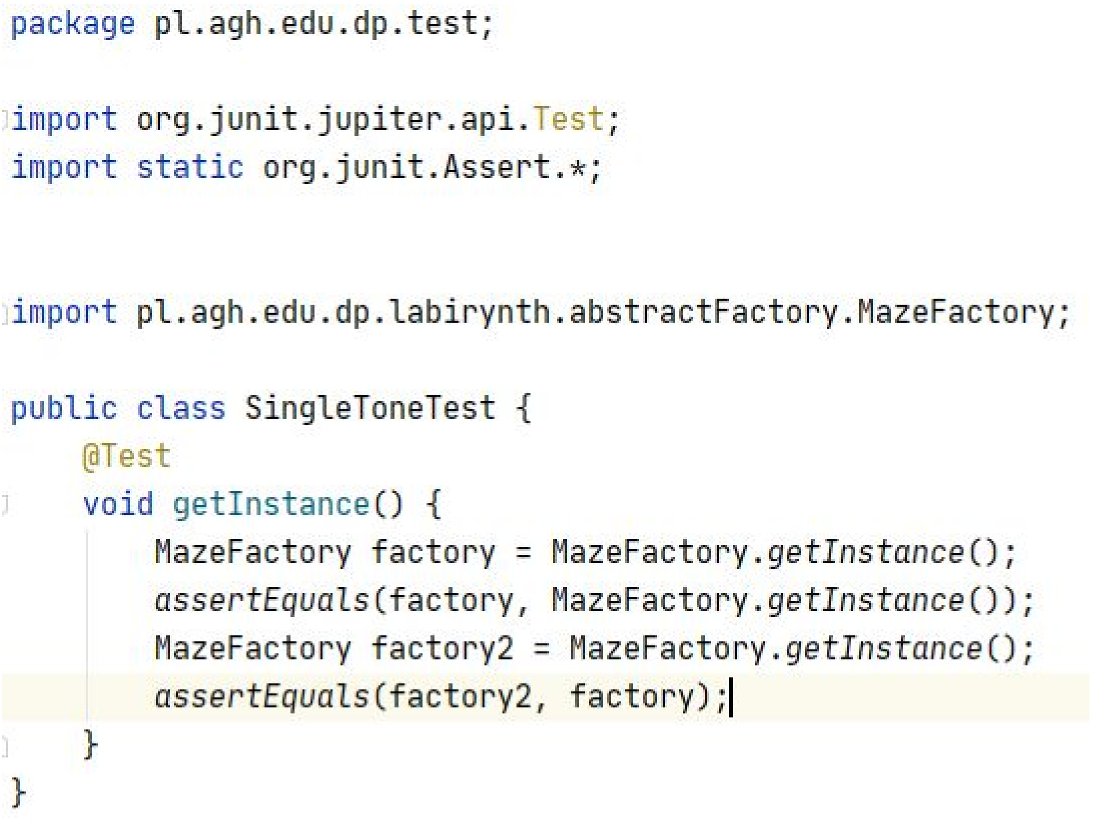
```java
public class Main {

    public static void main(String[] args) throws IOException {

        MazeGame mazeGame = new MazeGame();
        MazeFactory mazeFactory = MazeFactory.getInstance();
        CountingMazeBuilder builderC = new CountingMazeBuilder();
        StandardMazeBuilder builderS = new StandardMazeBuilder();
        mazeGame.createMaze(builderC, mazeFactory);
        mazeGame.createMaze(builderS, mazeFactory);

        Player player = new Player(builderS.getCurrentMaze());

        while (true){
            char input = (char) System.in.read();

            switch(input){
                case 'E': player.move(Direction.East); break;
                case 'W': player.move(Direction.West); break;
                case 'N': player.move(Direction.North); break;
                case 'S': player.move(Direction.South); break;
                case 'Z': System.exit( status: 0);
            }
        }
    }
}
```

Przykładowa gra

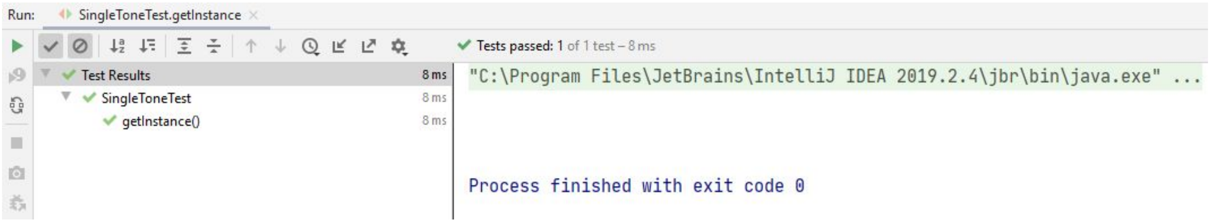


```
W
This is Wall
N
This is Wall
E
This is door
You've entered room
Moved from 1 to 2
S
This is Wall
W
This is door
You've entered room
Moved from 2 to 1
```

b) MazeFactory test singletona



```
package pl.agh.edu.dp.test;

import org.junit.jupiter.api.Test;
import static org.junit.Assert.*;


import pl.agh.edu.dp.labirynth.abstractFactory.MazeFactory;

public class SingleToneTest {
    @Test
    void getInstance() {
        MazeFactory factory = MazeFactory.getInstance();
        assertEquals(factory, MazeFactory.getInstance());
        MazeFactory factory2 = MazeFactory.getInstance();
        assertEquals(factory2, factory);
    }
}
```



```
Tests passed: 1 of 1 test – 8 ms

Test Results          8 ms
  SingleToneTest      8 ms
    getInstance()     8 ms
```

```
"C:\Program Files\JetBrains\IntelliJ IDEA 2019.2.4\jbr\bin\java.exe" ...


Process finished with exit code 0
```