Laboratorium 3
Adrian Wójciak, Lena Obertyńska
https://github.com/awojciak/po_lab_3

1.Builder
1.1. Interfejs MazeBuilder

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

public interface MazeBuilder {
   void addRoom(Room room);

   void addDoor(Room room1, Room room2) throws Exception;

   void addWallBetween(Direction dir, Room room1, Room room2);
}
```

1.2. Nowy parametr

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

1.3 Spostrzeżenie

Dzięki dokonanym zmianom tworzenie złożonych obiektów zostało wydzielone do innej klasy oraz możliwe jest w ramach procesu konstrukcji tworzenie różnych reprezentacji, nie powtarzając przy tym tego samego kodu.

1.4.Klasa StandardBuilderMaze

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

public class StandardBuilderMaze implements MazeBuilder {
   private Maze currentMaze;

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

   @Override
   public void addRoom(Room room) {
       room.setSide(Direction.North, new Wall());
       room.setSide(Direction.East, new Wall());
       room.setSide(Direction.South, new Wall());
       room.setSide(Direction.West, new Wall());
```

```
        this.currentMaze.addRoom(room);
    }

    private Direction commonWall(Room room1, Room room2) throws Exception {
        Direction common = null;

        for(Direction dir : Direction.values()) {
            if(room1.getSide(dir).equals(room2.getSide(dir.opposite()))) {
                common = dir;
            }
        }

        if(common == null) {
            throw new Exception("Nie można stworzyć drzwi");
        }

        return common;
    }

    @Override
    public void addDoor(Room room1, Room room2) throws Exception {
        Direction common = commonWall(room1, room2);

        Door door = new Door(room1, room2);
        room1.setSide(common, door);
        room2.setSide(common.opposite(), door);
    }

    @Override
    public void addWallBetween(Direction dir, Room room1, Room room2) {
        MapSite side = room1.getSide(dir);
        room2.setSide(dir.opposite(), side);
    }

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

Ponadto do enuma Direction dodano metodę opposite:

```
public Direction opposite() {
    switch (this) {
        case North:
            return South;
```

```
        case South:
            return North;
        case West:
            return East;
        default:
            return West;
    }
}
```

1.5. Tworzenie labiryntu

funkcja createMaze:

```
public Maze createMaze(StandardBuilderMaze builder) throws Exception {
    Room r1 = new Room(1);
    Room r2 = new Room(2);
    Room r3 = new Room(3);
    Room r4 = new Room(4);

    builder.addRoom(r1);
    builder.addRoom(r2);
    builder.addRoom(r3);
    builder.addRoom(r4);

    builder.addWallBetween(Direction.North, r1, r2);
    builder.addDoor(r1, r2);

    builder.addWallBetween(Direction.West, r2, r3);
    builder.addDoor(r2, r3);

    builder.addWallBetween(Direction.South, r3, r4);
    builder.addDoor(r3, r4);

    return builder.getCurrentMaze();
}
```

funkcja main:

```
public static void main(String[] args) throws Exception {
    MazeGame mazeGame = new MazeGame();
    StandardBuilderMaze builder = new StandardBuilderMaze();

    Maze maze = mazeGame.createMaze(builder);

    System.out.println(maze.getRoomNumbers());
```

```
}
```

Wynik: 4 (poprawny)

1.6. Klasa CountingMazeBuilder

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

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

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

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

   @Override
   public void addDoor(Room room1, Room room2) throws Exception {
       this.walls--;
       this.doors++;
   }

   @Override
   public void addWallBetween(Direction dir, Room room1, Room room2) {
       this.walls--;
   }

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

tymczasowa modyfikacja metody createMaze w celu testów:

```
public void createMaze(CountingMazeBuilder builder) throws Exception {
   Room r1 = new Room(1);
   Room r2 = new Room(2);
   Room r3 = new Room(3);
```

```
    Room r4 = new Room(4);

    builder.addRoom(r1);
    builder.addRoom(r2);
    builder.addRoom(r3);
    builder.addRoom(r4);

    builder.addWallBetween(Direction.North, r1, r2);
    builder.addDoor(r1, r2);

    builder.addWallBetween(Direction.West, r2, r3);
    builder.addDoor(r2, r3);

    builder.addWallBetween(Direction.South, r3, r4);
    builder.addDoor(r3, r4);

    builder.addWallBetween(Direction.East, r4, r1);
}
```

tymczasowa modyfikacja funkcji main:

```
    public static void main(String[] args) throws Exception {
        MazeGame mazeGame = new MazeGame();
//        StandardBuilderMaze builder = new StandardBuilderMaze();
        CountingMazeBuilder builder = new CountingMazeBuilder();

//        Maze maze = mazeGame.createMaze(builder);
//        System.out.println(maze.getRoomNumbers());

        mazeGame.createMaze(builder);
        System.out.println(builder.getCounts());
    }
```

Wynik: 16 (poprawny: 4 pokoje + 9 ścian + 3 drzwi)

2. Fabryka abstrakcyjna
2.1. Klasa MazeFactory

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

public class MazeFactory {
    public Room makeRoom(int num) {
        return new Room(num);
    }
```

```
    public Door makeDoor(Room room1, Room room2) {
        return new Door(room1, room2);
    }

    public Wall makeWall() {
        return new Wall();
    }
}
```

2.2. Modyfikacja funkcji createMaze

```
public Maze createMaze(StandardBuilderMaze builder, MazeFactory factory)
throws Exception {
    Room r1 = factory.makeRoom(1);
    Room r2 = factory.makeRoom(2);
    Room r3 = factory.makeRoom(3);
    Room r4 = factory.makeRoom(4);

    builder.addRoom(r1);
    builder.addRoom(r2);
    builder.addRoom(r3);
    builder.addRoom(r4);

    builder.addWallBetween(Direction.North, r1, r2);
    builder.addDoor(r1, r2);

    builder.addWallBetween(Direction.West, r2, r3);
    builder.addDoor(r2, r3);

    builder.addWallBetween(Direction.South, r3, r4);
    builder.addDoor(r3, r4);

    builder.addWallBetween(Direction.East, r4, r1);

    return builder.getCurrentMaze();
}
```

2.3. Klasa EnchantedMazeFactory

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

public class EnchantedMazeFactory extends MazeFactory {
    @Override
```

```
    public Room makeRoom(int num) {
        return new EnchantedRoom(num);
    }

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

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

Podklasy elementów labiryntu:

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

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

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

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

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

public class EnchantedDoor extends Door {
    public EnchantedDoor(Room r1, Room r2) {
        super(r1, r2);
    }
}
```

2.4. Klasa BombedMazeFactory

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

public class BombedMazeFactory extends MazeFactory {
   @Override
   public Room makeRoom(int num) {
       return new BombedRoom(num);
   }

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

Podklasy elementów labiryntu:

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

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

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

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

3. Singleton

Zmiany w MazeFactory.

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

   public Room makeRoom(int num) {
       return new Room(num);
   }
```

```
    public Door makeDoor(Room room1, Room room2) {
        return new Door(room1, room2);
    }

    public Wall makeWall() {
        return new Wall();
    }
}
```

4. Rozszerzenie aplikacji labirynt
Mechanizm przemieszczania się:

```
Do you want to start a game? (y/n)
>y


---------------------------

Print 'end' to finish the game
You have 15s.... GO!
********************************
**        door      **
**  wall    1    wall  **
**        wall        **
********************************
>TIME LEFT: 15  Where do you go? (w/a/s/d)
>w
********************************
**        wall      **
**  door    2    wall  **
**        door        **
********************************
>TIME LEFT: 12  Where do you go? (w/a/s/d)
>a
********************************
**        wall      **
**  wall    3    door  **
**        door        **
********************************
>TIME LEFT: 9  Where do you go? (w/a/s/d)
```

```
public class Player {
    public Room currentRoom;

    public Player(Room startRoom){   this.currentRoom=startRoom;    }

    public void goTo(Direction side){

this.currentRoom=currentRoom.getSide(side).Enter(this.currentRoom);
```

```
    }
}
```

Klasa MazeGame:

```
private void move(Player player){
    System.out.print("Where do you go? (w/a/s/d) \n>");
    Scanner scanner = new Scanner(System.in);
    String playerDesizion = scanner.nextLine();
    switch (playerDesizion){
        case ("w"):
            player.goTo(Direction.North);
            break;
        case ("a"):
            player.goTo(Direction.West);
            break;
        case ("d"):
            player.goTo(Direction.East);
            break;
        case ("s"):
            player.goTo(Direction.South);
            break;
    }
    System.out.print(player.currentRoom.toString()+"\n>");
}

public boolean boomGame(Player player){
    Instant start = Instant.now();
    String playerDecision="";
    System.out.print(player.currentRoom.toString()+"\n>");
    while (!playerDecision.equals("end") &&
!player.currentRoom.isLast()
            && Duration.between(start,Instant.now()).toSeconds()<15)
{
        System.out.print("TIME LEFT: "+
(15-(Duration.between(start,Instant.now()).toSeconds()))+"  ");
        move(player);
    }
    if(player.currentRoom.isLast()) return true;
    else {
        System.out.println("bom bom bomBOMBOM
BBBBBBBBBOOOOOOOOOOOOOOMMM!!!!!!!!!");
        return false;
    }
}
```

```
public boolean game(Player player){
    String playerDesizion="";
    System.out.print(player.currentRoom.toString()+"\n>");
    while (!playerDesizion.equals("end") &&
!player.currentRoom.isLast()) { move(player); }
    if(player.currentRoom.isLast()) return true;
    else return false;
}

public boolean startGame(Maze maze){
    Player player = new Player(maze.getStartRoom());
    if(player.currentRoom.timer) {
        System.out.println("You have 15s.... GO!");
        return boomGame(player);
    }
    else return game(player);
}
```

Czy MazeFactory to singleton?

```
MazeFactory factory1 = MazeFactory.getInstance();
MazeFactory factory2 = MazeFactory.getInstance();
System.out.println("SINGLETON?: "+factory1.equals(factory2));
```

```
Main > main()
Main
"C:\Program Files\Java\jdk-10.0.2\bin\java.exe" "-javaagent:C:\Program
SINGLETON?: true
```