How SOLID is our Mars Rover

Luzern, 22. October 2021

ALCOR Academy Training

SOLID Principles

- **S** ingle Responsibility
- **O** pen/Close
- **L** iskow Substitution
- **I** nterface Segregation
- **D** ependency Inversion

```
public class Rover {
   private Position position;
   private Map<RoverCommand, PositionCommand> positionCommandMap =…

   public Position getPosition() {
     return position;
   }
   public void executeRoverCommands(List<RoverCommand> roverCommands) {
      for (RoverCommand roverCommand : roverCommands) {
         position = positionCommandMap.get(roverCommand).execute(position);
      }
   }
}
```

```
public class Position {
   final Plateau plateau;
   final Coordinate coordinate;
   final Direction direction;
   private Map<Direction, CoordinateCommand> coordinateCommandMap = new HashMap<>();
  …
  public Position turnHalfLeft() {
    return new Position(coordinate, plateau, direction.turnHalfLeft());
  }
  ..
   Position move() {
      return new Position(coordinateCommandMap.get(direction).execute(coordinate), plateau, direction);
   }
```

Composition of behaviour with small independent methods

No «multi dot operations» besides the command pattern

Contract between classes

```
public Coordinate moveNorth(Coordinate threshold) {
  return new Coordinate(this.x, min(y + 1, threshold.y));
}
public Coordinate moveEast(Coordinate threshold) {
  return new Coordinate(min(x + 1, threshold.x), this.y);
}
```

Some refactoring beforehand..

Position, Coordinate --> immutable

```
public class Position {
    Direction direction;
    …
    public void turnLeft() {
        direction = direction.turnLeft();
    }
    public void turnRight() {
        direction = direction.turnRight();
    } …
```

```
public class Position {
    final Direction direction;
    …
    public Position turnLeft() {
        return new Position(coordinate, direction.turnLeft());
    }
    public Position turnRight() {
        return new Position(coordinate, direction.turnRight());
    } …
```

```
public class Coordinate {
    int x;
    int y;
    …
    public void moveNorth(Coordinate threshold) {
        y = min(y + 1, threshold.y);
    }
    public void moveEast(Coordinate threshold) {
        x = min(x + 1, threshold.x);
    } …
```

```
public class Coordinate {
    final int x;
    final int y;
    …
    public Coordinate moveNorth(Coordinate threshold) {
        return new Coordinate(this.x, min(y + 1, threshold.y));
    }
    public Coordinate moveEast(Coordinate threshold) {
        return new Coordinate(min(x + 1, threshold.x), this.y);
    } …
```

Some refactoring beforehand..

Rover, Position --> using Command Pattern

```
public class Rover {
    private Position position;
    …
    void executeRoverCommands(List<> roverCommands) {
        for (RoverCommand roverCommand : roverCommands) {
            if (roverCommand == TURN_LEFT) {
                position.turnLeft();
            }
            if (roverCommand == TURN_RIGHT) {
                position.turnRight();
            } …
        }
    }…
```

```
public class Rover {
    private Position position;
    private Map<RoverCommand, PositionCommand> positionCommandMap = new HashMap<>() {{
        put(TURN_LEFT, new PositionTurnLeftCommand());
        put(TURN_RIGHT, new PositionTurnRightCommand());
        …
    }};
    void executeRoverCommands(List<RoverCommand> roverCommands) {
        for (RoverCommand roverCommand : roverCommands) {
            position = positionCommandMap.get(roverCommand).execute(position);
        }
    } …
```

```
public class Position {
    final Coordinate coordinate;
    Direction direction;
    …
    void move(Coordinate topRightCorner) {
        if (direction == SOUTH) {
            coordinate.moveSouth();
        }
        if (direction == WEST) {
            coordinate.moveWest();
        }…
    }
```

```
public class Position {
    final Direction direction;
    Coordinate coordinate;
    private Map<Direction, CoordinateCommand> coordinateCommandMap =
            new HashMap<>() {{
        put(NORTH, new CoordinateMoveNorthCommand());
        put(SOUTH, new CoordinateMoveSouthCommand());
        …
    }};
    Position move() {
        return new Position(coordinateCommandMap.get(direction).execute(coordinate)
                                    , direction);
    }…
```

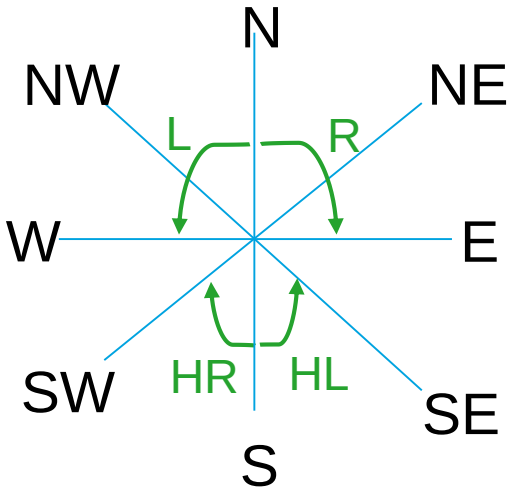
Additonal feature for the Mars Rover

In order to control a rover, NASA sends a simple string of letters. The possible letters are ‘L’, ‘R’ and ‘M’.
‘L’ and **‘R’** makes the rover spin **90 degrees** left or right respectively, without moving from its current spot.

Assume that the square directly **North** from (x, y) is (**x, y+1**).

In order to control a rover, NASA sends a simple string of letters. The possible letters are ‘L’, ‘HL’, ‘R’, ‘HR’ and ‘M’. …
‘HL’ and **‘HR’** makes the rover spin **45 degrees** left or right respectively, without moving from its current spot.

Assume that the square **North-East** from (x, y) is (x+1, y+1).



Feature Implementation

```
public enum Direction {
    NORTH(0),
    NORTHEAST(1),
    EAST(2),
    SOUTHEAST(3),
    SOUTH(4),
    SOUTHWEST(5),
    WEST(6),
    NORTHWEST(7);
    …
    Direction turnHalfLeft() { return … );}
    Direction turnHalfRight() { return …); }
}
```

```
public class Position {
    final Plateau plateau;
    private Map<Direction, CoordinateCommand> coordinateCommandMap = new HashMap<>();
    …
    public Position(Coordinate coordinate, Plateau plateau, Direction direction) {
        coordinateCommandMap.put(NORTH, new CoordinateMoveNorthCommand(plateau));
        …
        coordinateCommandMap.put(NORTHEAST, new CoordinateMoveNorthEastCommand(plateau));
        coordinateCommandMap.put(SOUTHEAST, new CoordinateMoveSouthEastCommand(plateau));
        coordinateCommandMap.put(NORTHWEST, new CoordinateMoveNorthWestCommand(plateau));
        coordinateCommandMap.put(SOUTHWEST, new CoordinateMoveSouthWestCommand(plateau));
    }
```

```
public class Rover {
    private Position position;
    private Map<RoverCommand, PositionCommand> positionCommandMap
            = new HashMap<>() {{ …
        put(TURN_HALF_LEFT,  new PositionTurnHalfLeftCommand());
        put(TURN_HALF_RIGHT, new PositionTurnHalfRightCommand());
    }};
```

```
public class PositionTurnLeftCommand implements PositionCommand {
    @Override
    public Position execute(Position position) {
        return position.turnHalfLeft().turnHalfLeft();
    }
}
```

```
public class CoordinateMoveNorthWestCommand implements CoordinateCommand {
    CoordinateMoveNorthWestCommand(Plateau plateau) {
        this.topRightCorner = plateau.topRightCorner;
        this.bottomLeftCorner = plateau.bottomLeftCorner;
    }
    @Override
    public Coordinate execute(Coordinate coordinate) {
        return coordinate.moveNorth(this.topRightCorner).moveWest(this.bottomLeftCorner);
    }
}
```

Open/Close Principle at the Mars Rover

- Strong cohesion and single responsiblity in the Rover and Position classes enables command pattern
- All class fields are used by all methods of the classes
- Methods for existing behaviour can be combined to implement new behaviour
- Immutable classes enables method concatenation
- All methods are at the same level of abstraction
- No inheritance is needed

Thank you for attention…

Any questions ?

Possible discussion aspects are…

- Is code using command pattern still obvious or maybe overengineered (yagni) ?
- Does immutable classes violates command query segregation ?

Contact: roderich.gross@css.ch