**Robot Controller Documentation**

## Core Components

### 1. Program Class

**Purpose**: Main entry point and user interface

* Handles user input for:
  + **Grid dimensions**
  + **Robot's starting position and direction**
  + **Movement commands**
* Performs **input validation**
* Sets up **sample obstacles**
* Displays **final position and traversal path**

### 2. Position Record

**Purpose**: Represents a location on the grid

* **Properties**:
  + **X**: X-coordinate (horizontal position)
  + **Y**: Y-coordinate (vertical position)
* Implemented as a **record type** for immutability
* Custom **ToString()** for formatted output: **(X, Y)**

### 3. Grid Class

**Purpose**: Manages the game board and obstacles

* **Properties**:
  + **Width**: Grid width
  + **Height**: Grid height
  + **\_obstacles**: Dictionary storing obstacle positions
* **Key Methods**:
  + **AddObstacle()**: Places obstacles on the grid
  + **IsValidPosition()**: Checks if a position is within bounds
  + **TryGetObstacle()**: Attempts to retrieve an obstacle at a position

### 4. Robot Class

**Purpose**: Controls robot movement and state

* **Properties**:
  + **Position**: Current location
  + **Facing**: Current direction
  + **Path**: List of traversed positions
* **Key Methods**:
  + **ExecuteCommands()**: Processes movement commands
  + **TurnLeft() / TurnRight()**: Handles rotation
  + **MoveForward()**: Handles forward movement
  + **HandleObstacle()**: Processes obstacle interactions

### 5. Obstacles

#### IObstacle Interface

**Purpose**: Base contract for all obstacles

* **Method**: AffectRobot(Robot robot)

#### Rock Class

* **Blocks robot movement**
* No effect implementation needed

#### Hole Class

* **Teleports** robot to a connected location
* Maintains **exit position**
* Updates **robot position** when encountered

#### Spinner Class

* **Rotates** robot by specified degrees
* Validates rotation in **90-degree increments**
* Updates **robot direction** when encountered

### 6. Direction Enum

* **North (0)**
* **East (1)**
* **South (2)**
* **West (3)**

## Movement Rules

### 1. Basic Movement

* **F**: Move forward one space
* **L**: Rotate 90 degrees left
* **R**: Rotate 90 degrees right

### 2. Boundary Handling

* **Robot cannot move outside grid boundaries**
* **Attempted boundary crossings are logged and ignored**

### 3. Obstacle Interactions

* **Rock**: Blocks movement
* **Hole**: Teleports to exit position
* **Spinner**: Rotates robot by specified degrees
* **Unknown obstacles**: Block movement

## Path Tracking

* Records **initial position**
* Adds **new positions after successful moves**
* Records **positions after teleportation**
* Records **positions after spinner effects**

## Error Handling

### 1. Input Validation

* **Grid dimensions must be positive**
* **Start position must be within grid**
* **Commands must be L, R, or F**
* **Obstacle positions must be valid**

### 2. Runtime Checks

* **Boundary validation**
* **Obstacle detection**
* **Direction validation**
* **Null checks for required parameters**

## Usage Example

Enter grid size (Width Height):

5 5

Enter robot start position (X Y Direction):

1 2 North

Enter movement commands (e.g., LFFFRFFL):

LFFFRFFL

Final Position: (0, 4) Facing: West

Path Traversed:

(1, 2)

(0, 2)

(0, 3)

(0, 4)

## Extension Points

### 1. New Obstacles

* Implement IObstacle interface
* Add obstacle handling in **Robot** class
* Register obstacle in **grid setup**

### 2. New Robot Types

* Implement IRobot interface (could be added)
* Customize **obstacle interactions**
* Add **specialized movement patterns**

### 3. Additional Commands

* Extend **command parsing** in ExecuteCommand()
* Add **new movement methods**
* Update **validation logic**

## Performance Considerations

* **Movement delay** of **100ms** for visualization
* **Dictionary** used for efficient **obstacle lookup**
* **Path stored as List** for sequential access
* **Immutable Position record** for thread safety