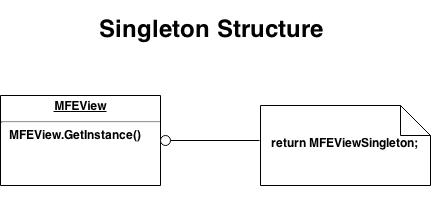
**PROJECT-3-DOCUMENTATION**

**STUDENT NAME: DINESH KASU (U00747253) EMAIL:** [**KASU.3@WRIGHT.EDU**](mailto:KASU.3@WRIGHT.EDU)

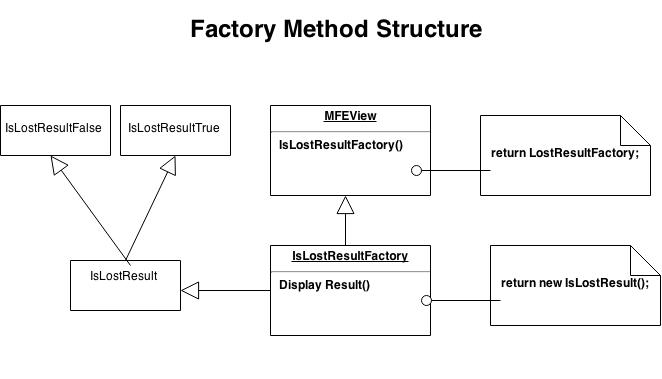


**Participants:**

* MFEView: MFEView defines an Instance operation that lets MFEController access its unique instance.

**Collaboration:**

* MFE Controller access a Singleton instance solely through MFE View’s Instance operation.



**Participants:**

**IsLossResult:**

* Defines the interface of objects the factory method isLossResultFactory creates.

**IsLossResultTrue:**

* Implements the **IsLossResult** interface.

**IsLossResultFalse:**

* Implements the **IsLossResult** interface.

**MFEView:**

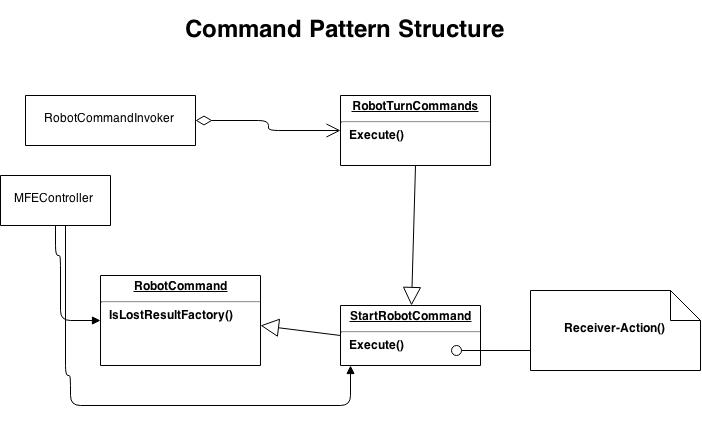
* Declares the factory method, which returns an object of type IsLossResult.
* MFEView may also define a default implementation of the factory method that returns a default IsLossResult object.
* may call the factory method to create a Product object.

**IsLossResultFactory:**

* Overrides the factory method to return an instance of a IsLossResult.

**Collaborations:**

* MFEView relies on its subclasses to define the factory method so that it returns an instance of the appropriate IsLossResult.



**Participants:**

**RobotTurnCommands**

* Declares an interface for executing an operation.

**StartRobotCommand**

* Defines a binding between a Receiver object RobotCommand and an action.implements Execute by invoking the corresponding operation(s) on Receiver.

**MFEController**

* Creates aStartRobotCommand object and sets its receiver.

**RobotCommandInvoker**

* Asks the Robotcommands to carry out the request.

**RobotCommand**

* Knows how to perform the operations associated with carrying out a request. Any class may serve as a Receiver.

**Collaboration:**

* The MFEController creates a RobotCommand object and specifies its receiver.
* An Invoker object stores the StartRobotCommand object.
* The invoker issues a request by calling Execute on the command. When commands are undoable,
* StartRobotCommand stores state for undoing the command prior to invoking Execute.
* The StartRobotCommand object invokes operations on its receiver to carry out the request.
* The following diagram shows the interactions between these objects. It illustrates how Command decouples





**Model View Controller Design Pattern:-**

The Mutant Flat world explorer problem implementation is divided into model, view, & controller

modules.

**1. Model module Responsibilities:**

* Model module contains the information about the robot grid positions and its orientation being
* processed.
* It takes the scanned input file data from MFEController and loads the information into variables.
* It modifies the grid positions of the robot and its orientation based on the instructions.
* It also set the scent for the grid positions when the robot fall of from the boundaries.
* This module consists of the following classes.

RobotModel.java RobotCommand.java, StartRobotCommand.java, RobotTurnCommand.java,

RobotCommandInvoker.java

**2. View module Responsibilities:**

* View module displays the final grid position and orientation of robot.
* If a robot falls off the edge of the grid, then it also displays 'LOST' after grid position and
* orientation of robot.
* This module consists of the following classes.

MFEView.java, IsLostResultResult.java, IsLostFalse.java,IsLostResultFactory.java ,IsLostTrue.java

**3. Controller module Responsibilities:**

* This module is the main entry point for Mutant Flat world Explorers.
* Creates the instance of new robot model.
* Tells the view to print the output when the robot instructions are over.
* This module consists of the following class.

MFEController.java

**Accessors:-**

An accessor (getter) method is used to return the value of a private field. These methods always return the same data type as their corresponding private field and then simply return the value of that private field.These are used in **RobotModel.java** program.

**Singleton:-**

* In order to make a class Singleton, we have to follow the below rules:
* It should have only single instance
* This instance should be available through a global access

**Steps to make class as a singleton.**

1. Create a class which you want to make singleton.
2. Create a private default constructor of the class.
3. Create a private static variable of the class created in step1 and this variable should be private and
4. static and it should refer to the instance of class created in step1.
5. Create an accessor method which could always return us back with an instance of class created in
6. step3.
7. There should not be any method or constructor which can create instance of this class.
8. Singleton Design pattern is applied to **MFEView.java** class, as it can have only one single instance in the
9. game.

**Factory Method:-**

* This pattern is used to find the isLost result of robot on the grid: True or False.
* I have created an **IsLostResult** interface and concrete classes-**IsLostTrue** & **IsLostFalse**
* implementing the **IsLostResult** interface. A factory class **IsLostResultFactory** is defined as a next step.
* **MFEView**, our driver class will use **IsLostResultFactory** to get an **IsLostResult** object. The MFEView
* will send the IsLost information to **IsLostResultFactory** to get the type of object it needs.

**Command Pattern:-**

* This pattern is used to process the Robot orientation and instructions in the model module. I have
* created an interface **StartRobotCommand** which is acting as a command. I have created
* a **RobotCommand** class which acts as a request. I have concrete command
* classes **RobotTurnCommand** implementing StartRobotCommand interface which will do actual
* command processing. A class **RobotCommandInvoker** is created which acts as a invoker object. It can
* take steps and process steps. **RobotCommandInvoker** object uses command pattern to identify which
* object will execute which command based on type of command. **MFEController**, our driver class will use **RobotCommandInvoker** class to demonstrate command pattern.

**Justification:-**

* I have chosen the model-view-controller design pattern and then implemented the singleton, factory method, accessories and command patterns. MVC (Model, View, and Controller) is a design pattern for organizing code in an application to improve maintainability.
* Here the Robot is the model, the input/interface is the controller and the grid is the view. Because the grid does not know about the robot. It is completely independent. This separation allows the controller to load the robot model and make changes in the view. Also, used the singleton, factory method, accessories and command patterns.
* So, this all modules leading to loose coupling and also, we can use it for reusing purpose in other board related projects.