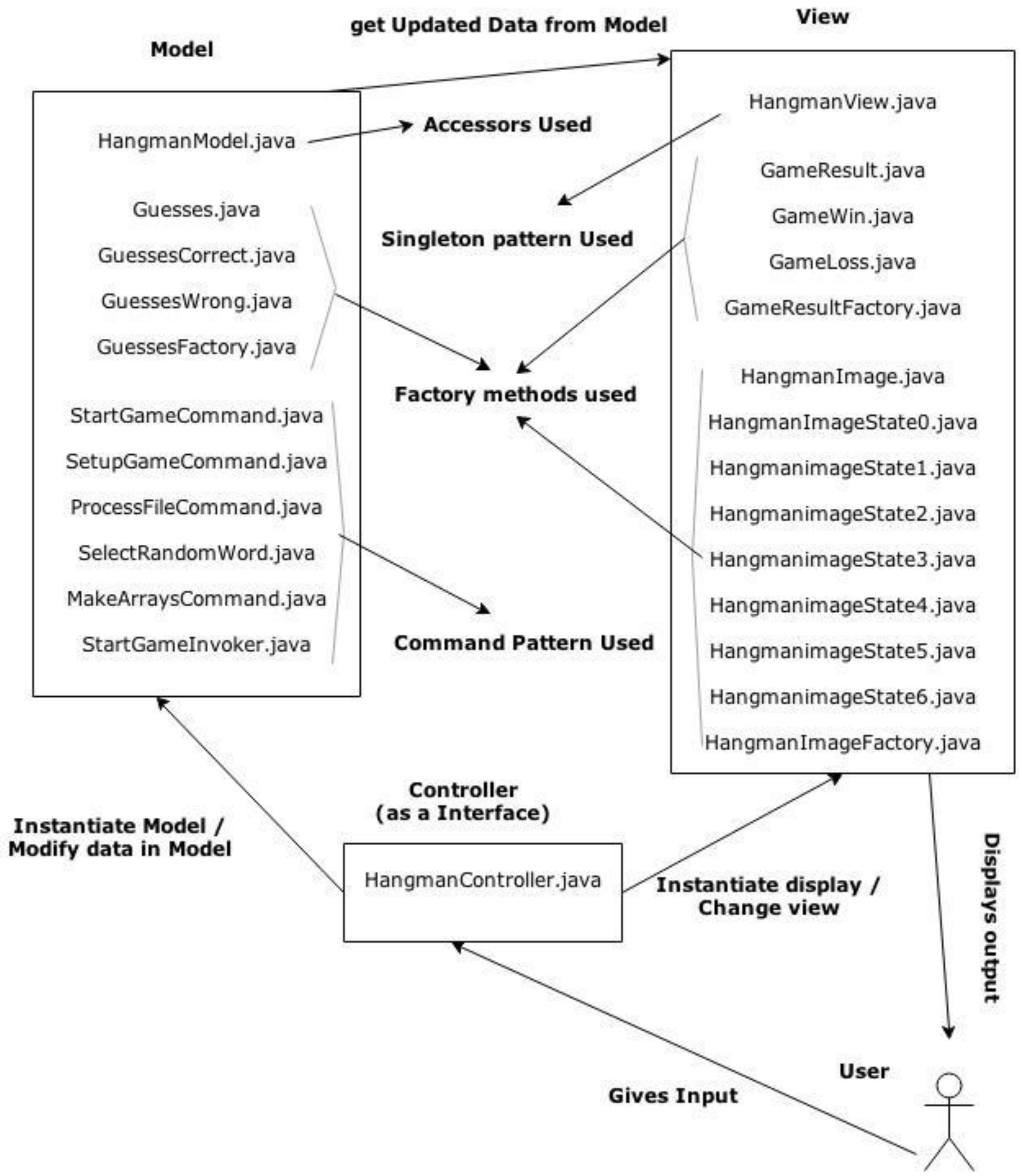
PROJECT-1-DOCUMENTATION

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

HANGMAN GAME IMPLEMENTATION USING DESIGN PATTERNS



<u>Model View Controller Design Pattern:-</u>

The game implementation is divided into model, view,& controller modules.

1.Model module Responsibilities:

- Model module contains the information about the word being processed.
- It scans the given input file, load the words into array and selects the random word
- It tracks the incorrect guess and correct guesses
- It modifies the positions of dashed word with a matching letter, when the same is guessed by user.
- Tells the view where to put letters
- This module consists of the following classes.

HangmanModel.java	StartGameCommand.java
	SetupGameCommand.java
Guesses.java	ProcessFilecommand.java
GuessesWrong.java	SelectRandomWord.java
GuessesCorrect.java	MakeArraysCommand.java
GuessesFactory.java	StartGameInvoker.java

2.View module Responsibilities:

- View module displays the results of module
- It draws the hangman states
- It displays the guessed letters, remaining choices and , game results-win or lose.
- This module consists of the following classes.

HangmanView.java	HangmanImage.java	HangmanimageState5.java
	HangmanimageState0.java	HangmanimageState6.java
GameResult.java	HangmanimageState1.java	HangmanImageFactory.java
GameWin.java	HangmanimageState2.java	
GameLose.java	HangmanimageState3.java	
GameResultFactory.java	HangmanimageState4.java	

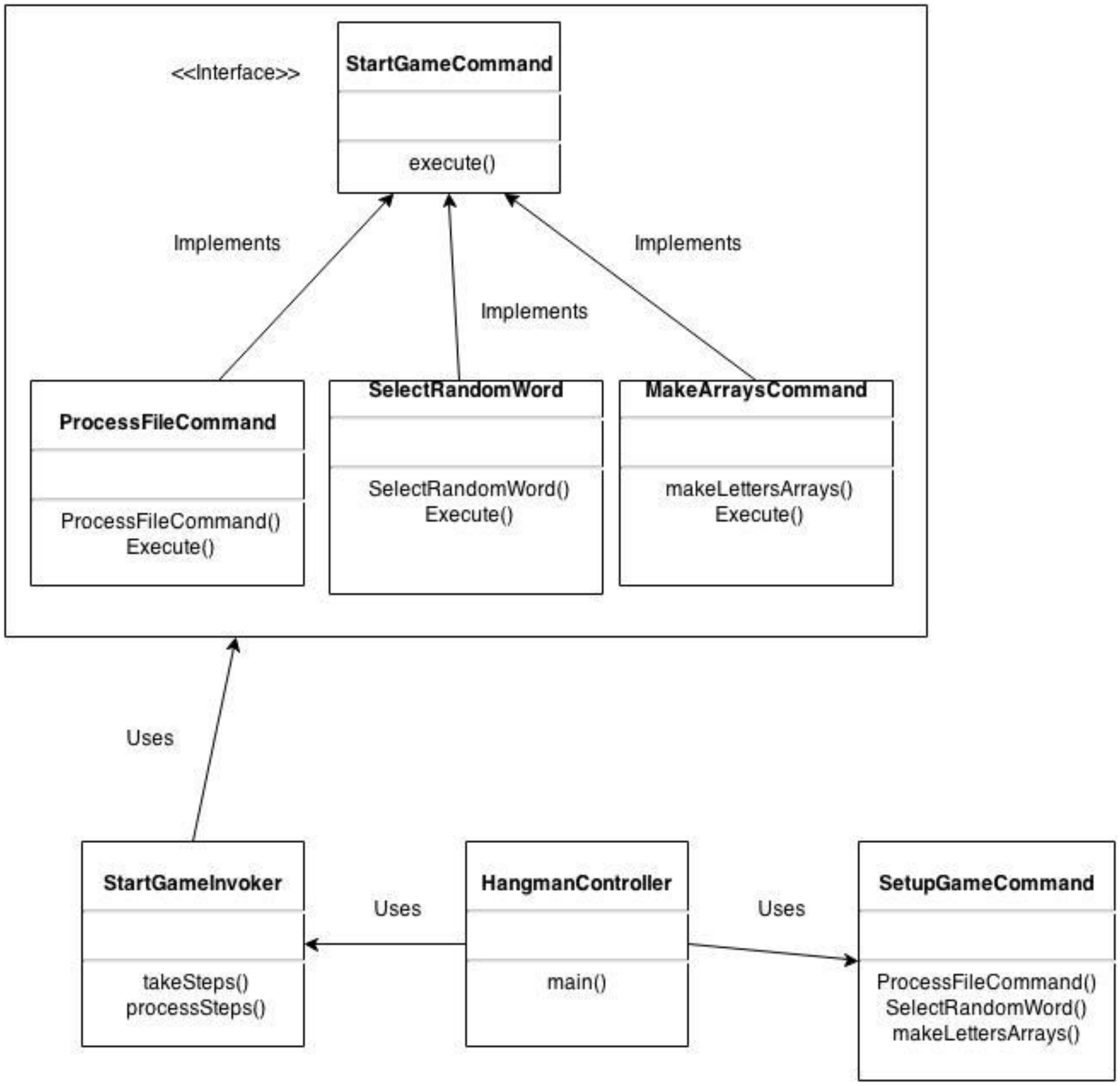
3.Controller module Responsibilities:

- This module is the main entry point for starting the game.
- Creates the instance of new game (Model).
- Tells the view when the game is over and exits the game.
- Tells the view when another game is going to start.
- This module consists of the following class.

HangmanController.java

<u>Command Pattern:-</u>

This pattern is used to startup the hangman game in the model module. I have created an interface **StartGameCommand** which is acting as a command. I have created a **SetupGameCommand** class which acts as a request. I have concrete command classes **ProcessFileCommand**,**SelectRandomWord** and **MakeArraysCommand** implementing StartGameCommand interface which will do actual command processing. A class **StartGameInvoker** is created which acts as a invoker object. It can take steps and process steps. **StartGameInvoker** object uses command pattern to identify which object will execute which command based on type of command. **HangmanController**, our driver class will use **StartGameInvoker** class to demonstrate command pattern.



<u>Accessors:-</u>

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 **HangmanModel.java** program.

<u>Singleton :-</u>

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 static and it should refer to the instance of class created in step1.
4. Create a accessor method which could always return us back with a instance of class created in step3.
5. There should not be any method or constructor which can create instance of this class.

Singleton Design pattern is applied to **HangmanView.java** class, as it can have only one single instance in the game.

<u>Factory Methods:-</u>The factory methods are used in the following 3 cases

1) This pattern is used when the user input: guess is matched with the letters present in the correct word, and then we can decide whether the guess is to be added to correct guess list or incorrect guess list.

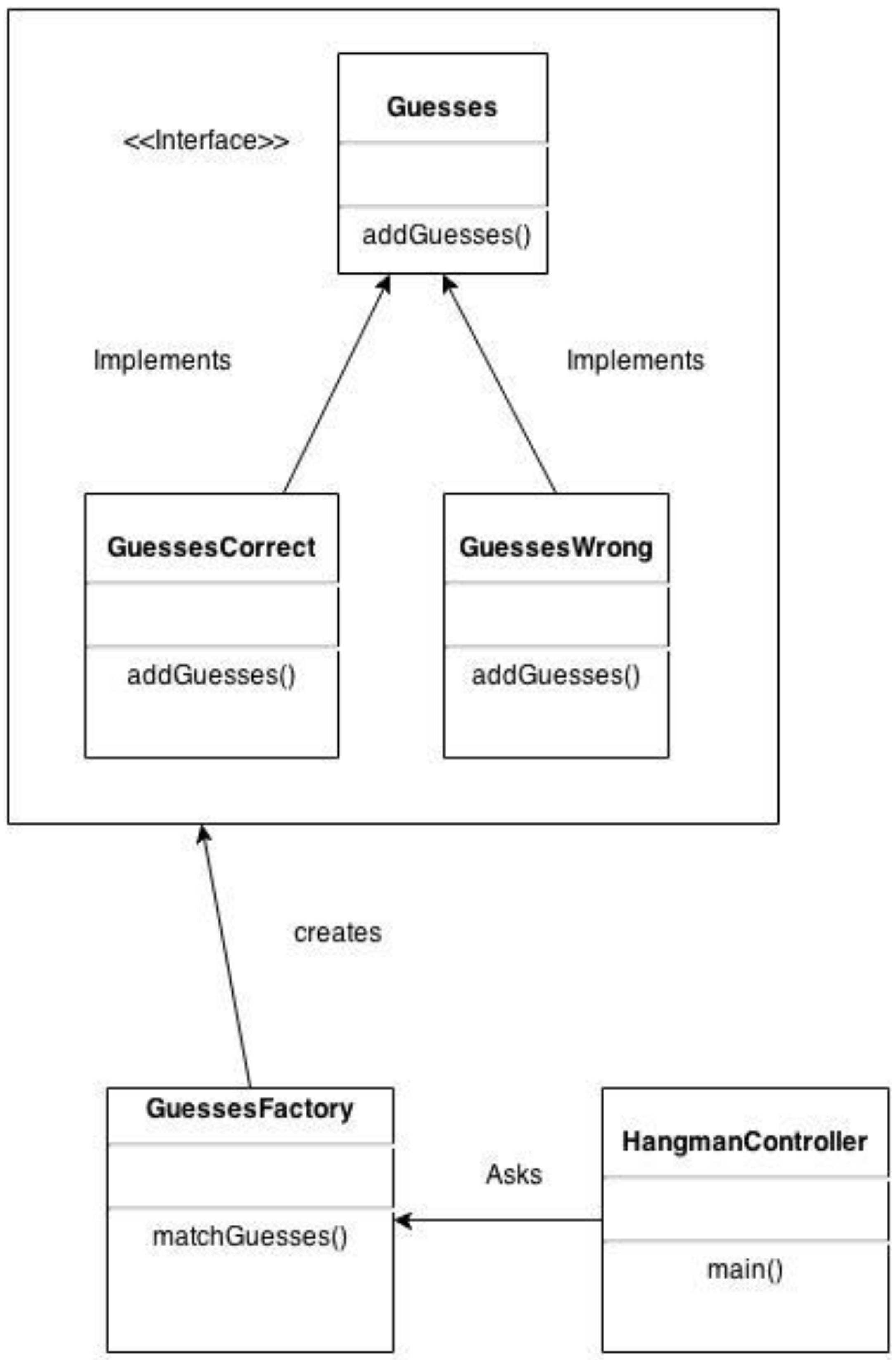
I have created a **Guesses** interface and concrete classes-**GuessesCorrect** & **GuessesWrong** implementing the **Guesses** interface. A factory class **GuessesFactory** is defined as a next step. **HangmanController**, our driver class will use **GuessesFactory** to get a **Guesses** object. It will pass information (**GuessesCorrect**/ **GuessesWrong**) to **GuessesFactory** to get the type of object it needs.

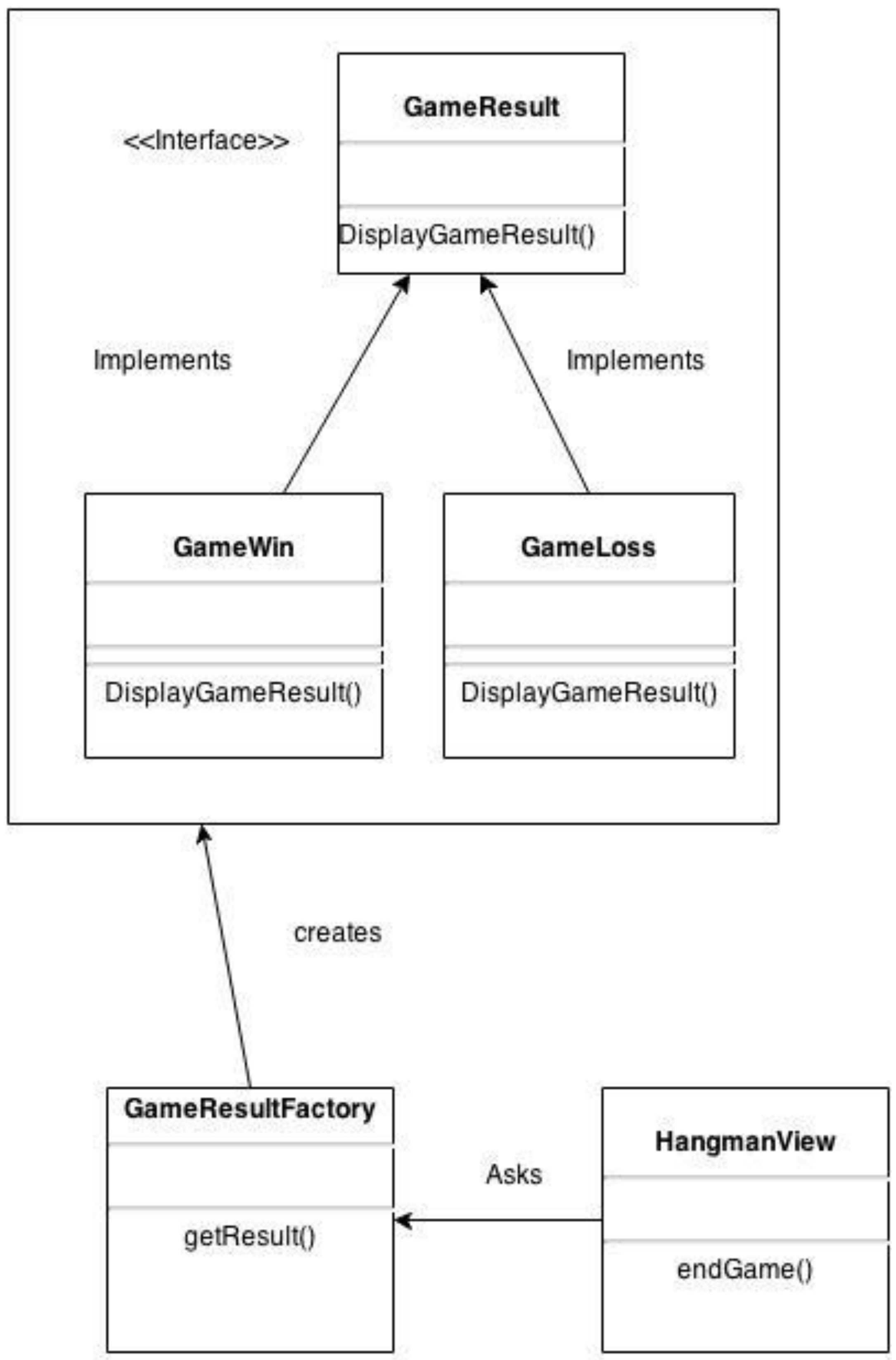
2) This pattern is used to find the Hangman game result :Won or Lose.

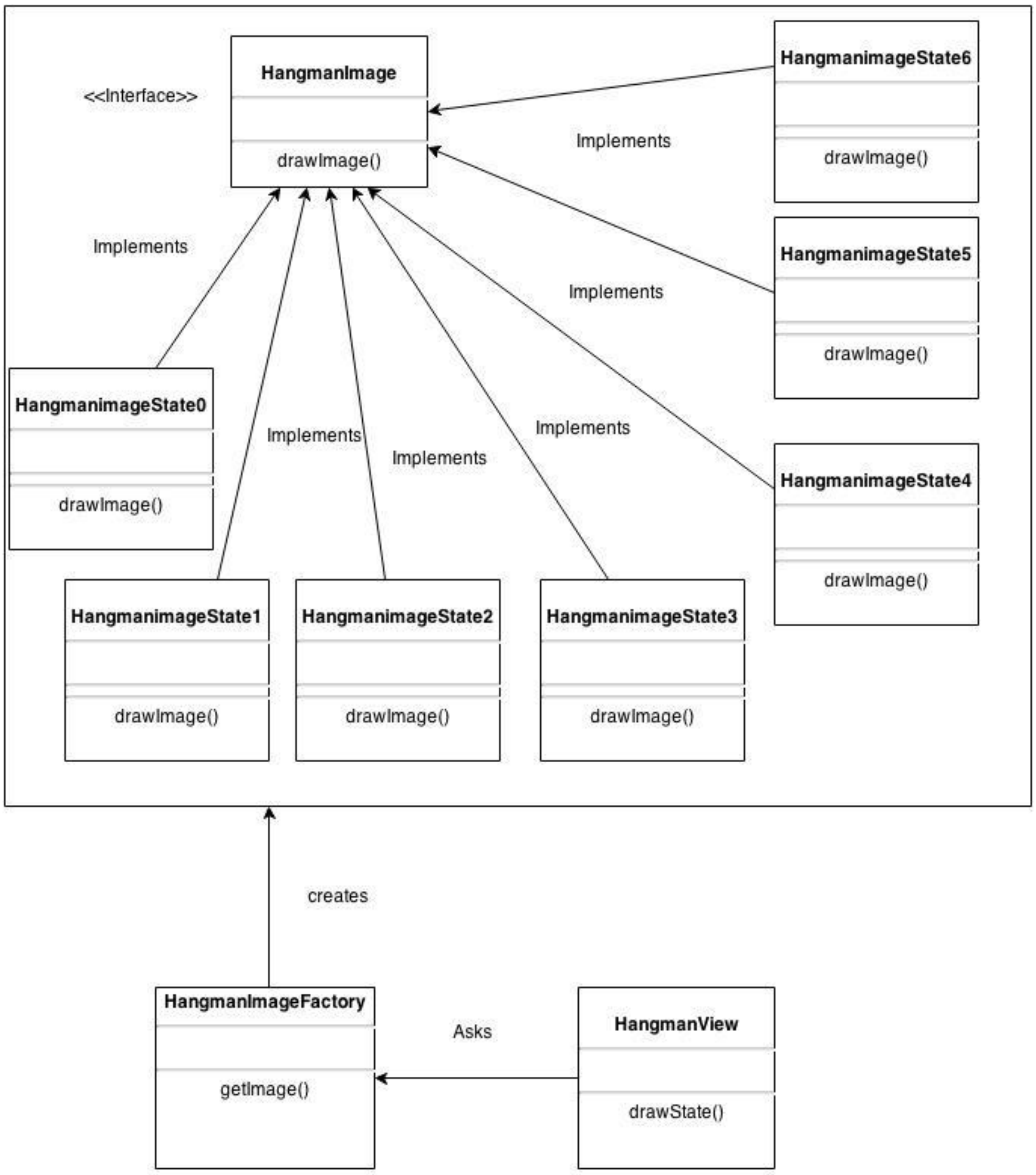
I have created a **GameResult** interface and concrete classes-**GameWin** & **GameLoss** implementing the **GameResult** interface. A factory class **GameResultFactory** is defined as a next step. **HangmanView**, our driver class will use **GameResultFactory** to get a **GameResult** object. **HangmanController** will pass information(**Won/Lose**) to the **HangmanView**. Then **HangmanView** will send the same information to **GameResultFactory** to get the type of object it needs.

3) This pattern is to display the hangman state images based on the bad guesses.

I have created a **Hangmanimage** interface and concrete classes(hangmanimagestates) implementing the **Hangmanimage** interface. A factory class **HangmanImageFactory** is defined as a next step. HangmanView, our view module will use **HangmanImageFactory** to get a **Hangmanimage** object. It will pass information (states indicator) to **HangmanImageFactory** to get the type of object it needs.


Guesses
<<Interface>>
addGuesses()
Implements
Implements
GuessesCorrect
addGuesses()
GuessesWrong
addGuesses()
creates
GuessesFactory
matchGuesses()
Asks
HangmanController
main()


GameResult
<<Interface>>
DisplayGameResult()
Implements
Implements
GameWin
DisplayGameResult()
GameLoss
DisplayGameResult()
creates
GameResultFactory
getResult()
Asks
HangmanView
endGame()


HangmanImage
<<Interface>>
drawImage()
Implements
HangmanimageState0
HangmanimageState1
HangmanimageState2
HangmanimageState3
HangmanimageState4
HangmanimageState5
HangmanimageState6
drawImage()
creates
HangmanImageFactory
getImage()
Asks
HangmanView
drawState()