**PLATE OF FIRE**

By

Prasanth Bhagavatula

A20355611

Submitted to

Omar Aldawud

Contents

[Design and Rules of Plate of Fire 3](#_Toc436994349)

[Plate of Fire Prototype 4](#_Toc436994350)

[Design Patterns 5](#_Toc436994351)

[Creational Design Patterns 5](#_Toc436994352)

[Singleton Design Pattern 5](#_Toc436994353)

[Abstract Factory Design Pattern 5](#_Toc436994354)

[Structural Design Patterns 6](#_Toc436994355)

[Decorator Design Pattern 6](#_Toc436994356)

[Composite Design Pattern 6](#_Toc436994357)

[Behavioural Design Patterns 7](#_Toc436994358)

[Strategy Design Pattern 7](#_Toc436994359)

[Command Design Pattern 7](#_Toc436994360)

[Final Class Diagram 8](#_Toc436994361)

[State Diagrams 9](#_Toc436994362)

[Hero State Diagram 9](#_Toc436994363)

[HeroFactory State Diagram 9](#_Toc436994364)

[DisguisedHero State Diagram 10](#_Toc436994365)

[Arena State Diagram 10](#_Toc436994366)

[FightStrategy State Diagram 11](#_Toc436994367)

[PowerUp State Diagram 11](#_Toc436994368)

[Code Snippets 12](#_Toc436994369)

[Singleton Design Pattern Code 12](#_Toc436994370)

[Abstract Factory Design Pattern 13](#_Toc436994371)

[Decorator Design Pattern 14](#_Toc436994372)

[Composite Design Pattern 15](#_Toc436994373)

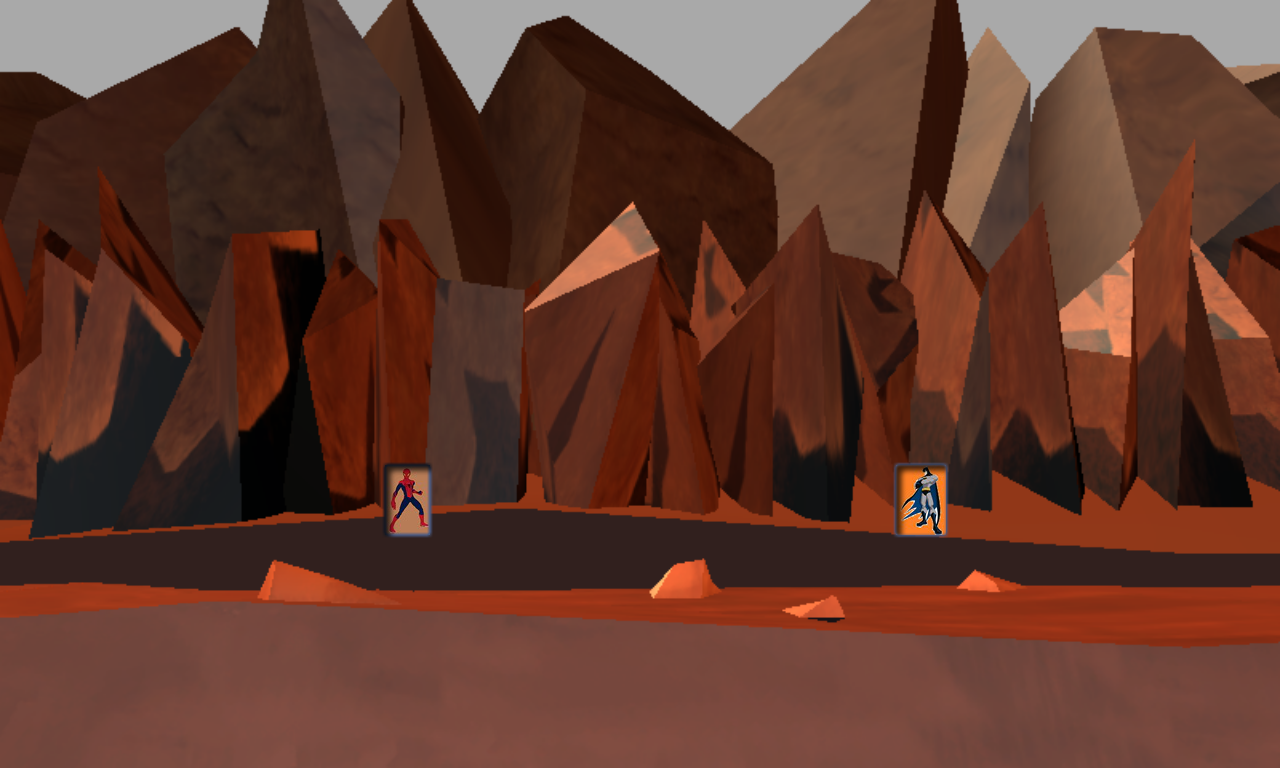
[Strategy Design Pattern 16](#_Toc436994374)

[Command Design Pattern 16](#_Toc436994375)

# Design and Rules of Plate of Fire

1. The game takes place in an “arena” which itself contains other objects. Other objects include mountains, canyons and the super heroes.
2. A player can select only one of the following three super heroes:
   1. Batman
   2. Spiderman
   3. Superman
3. Only two players can play the game at a time and both the players cannot select the same super hero.
4. The players are given an option to decorate their super heroes with either cape or cloak.
5. Each and every super hero has to choose a fighting strategy.
6. There are three fighting strategies.
   1. Kick Boxing
   2. Martial Arts
   3. Swords
7. The chosen super heroes must be of the same level.
8. Before the fight starts, each of the super heroes will be assigned a pre-defined power level.
9. As the fight progresses, the power level decreases based on the number of hits the other player makes.
10. During the fight, a super hero can grab power ups on the arena and using the power up panel, they can use the grabbed power up on the super hero.
11. Grabbing the power ups will also increase the power level of the super hero.
12. Each super hero has a special power and that can be used only once in the fight.
13. When the fight ends, the winning super hero power level will be increased and will be used for the next fight.
14. Once the power level reaches 100, the power level will be reset to 50 and the level will be increased.

# Plate of Fire Prototype



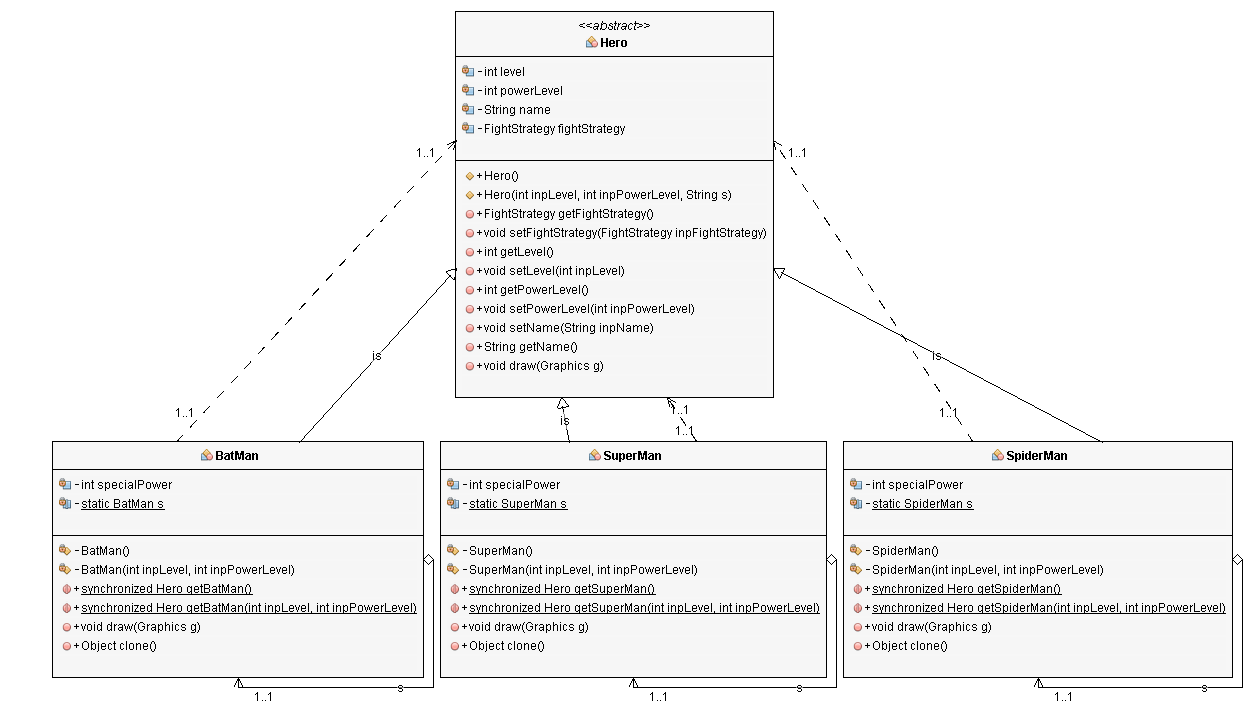
# Design Patterns

Two design patterns from each of the categories have been used to design the game.

## Creational Design Patterns

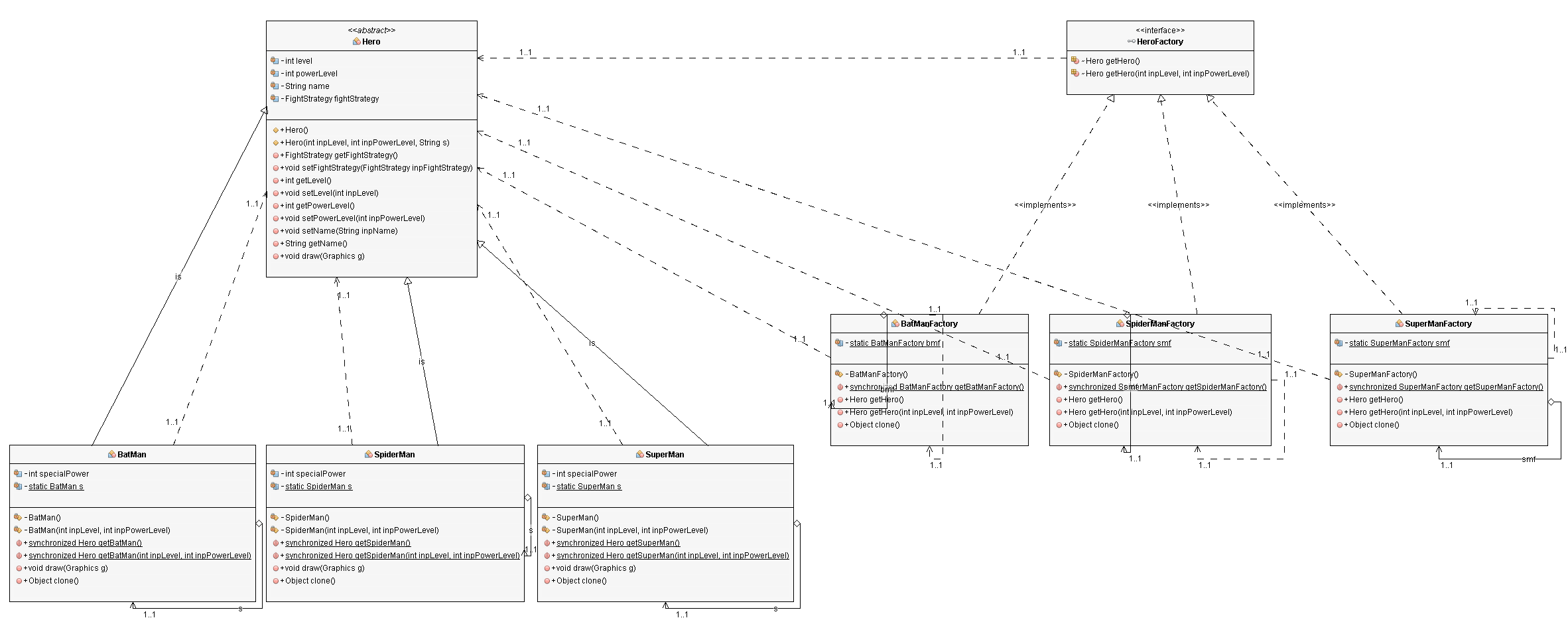
### Singleton Design Pattern

* This pattern is used to restrict the players creating more than a single object of a super hero.



### Abstract Factory Design Pattern

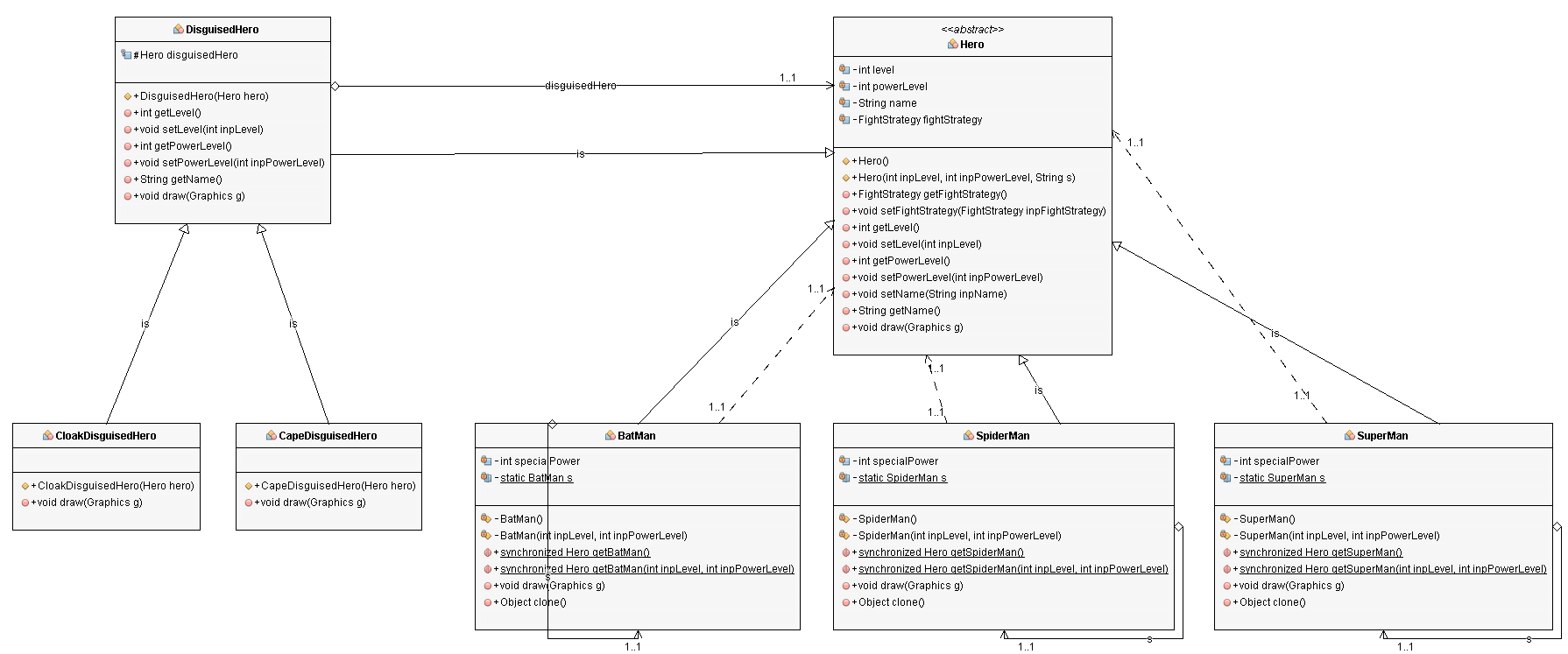
* This design pattern is used to create the super heroes.



## Structural Design Patterns

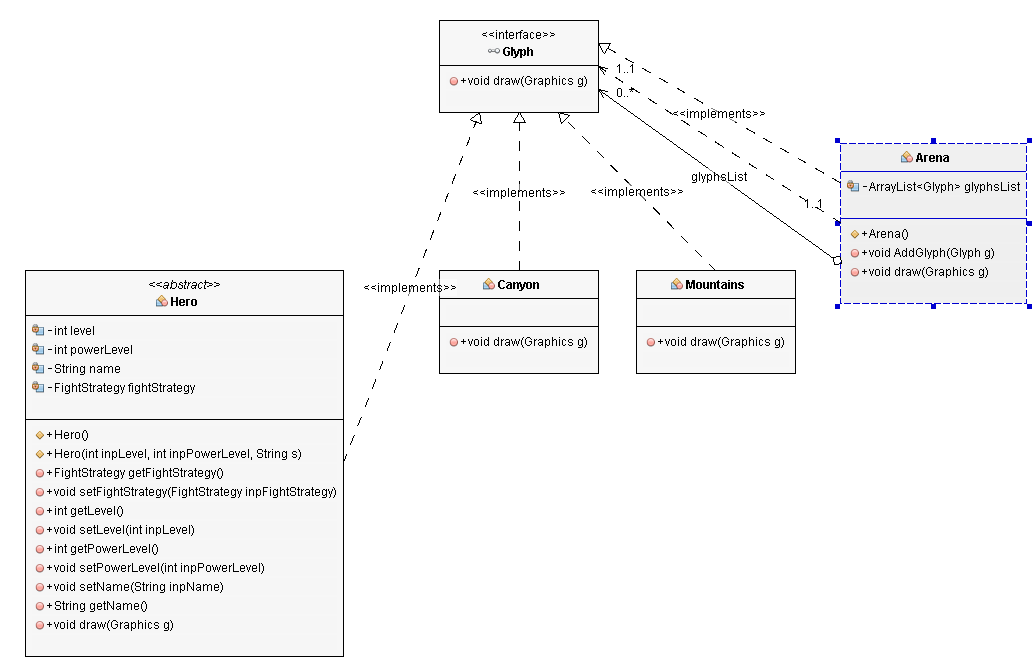
### Decorator Design Pattern

* This design pattern is used to decorate the super hero with either cape or cloak.



### Composite Design Pattern

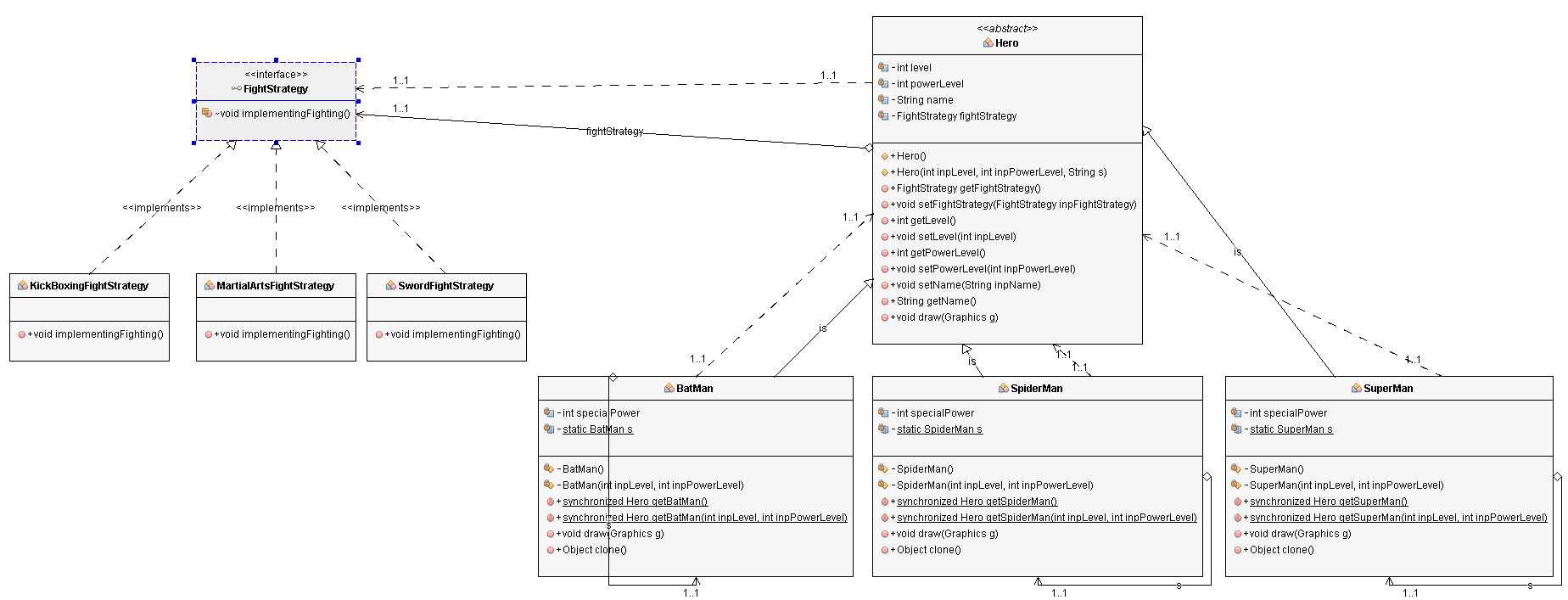
* This design pattern is used while designing the arena. All the objects including the arena implement an interface “Glyph”. The arena will have an list of glyphs that are created during the game.



## Behavioural Design Patterns

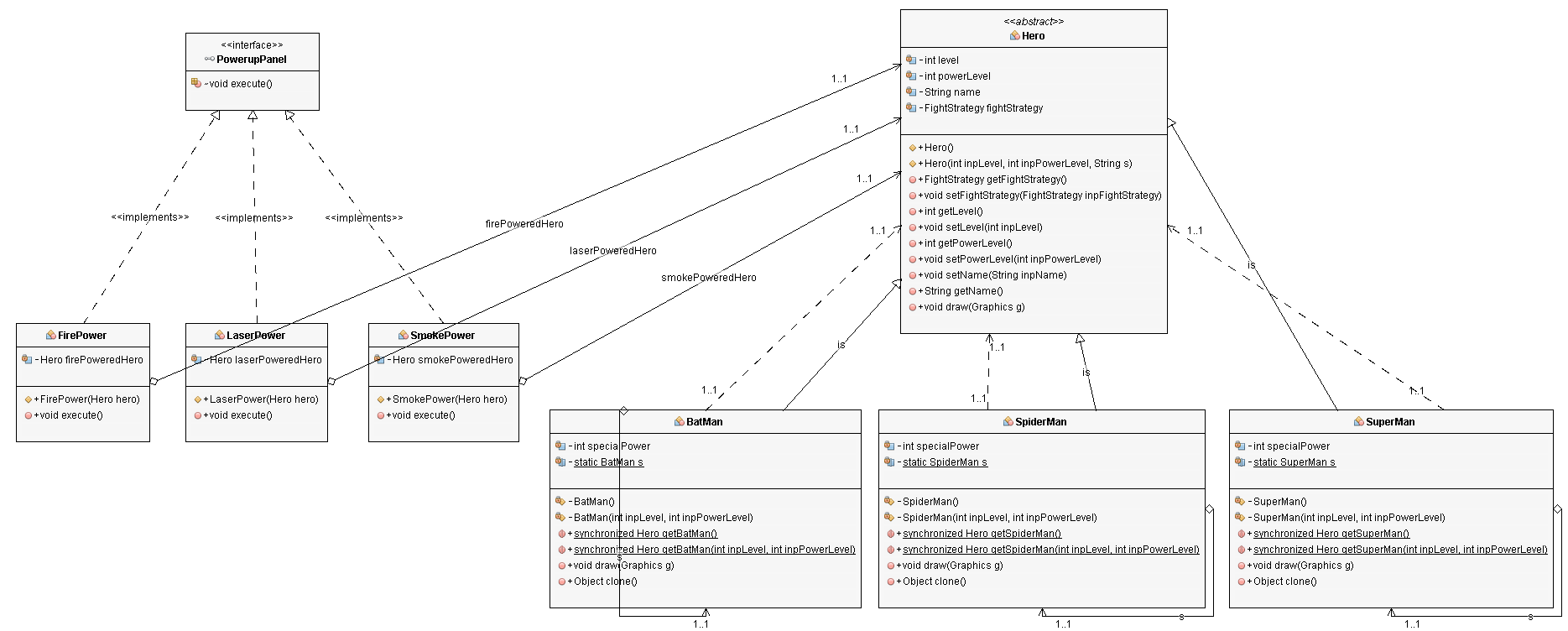
### Strategy Design Pattern

* This design pattern is used when the players need to choose a strategy for the fight.

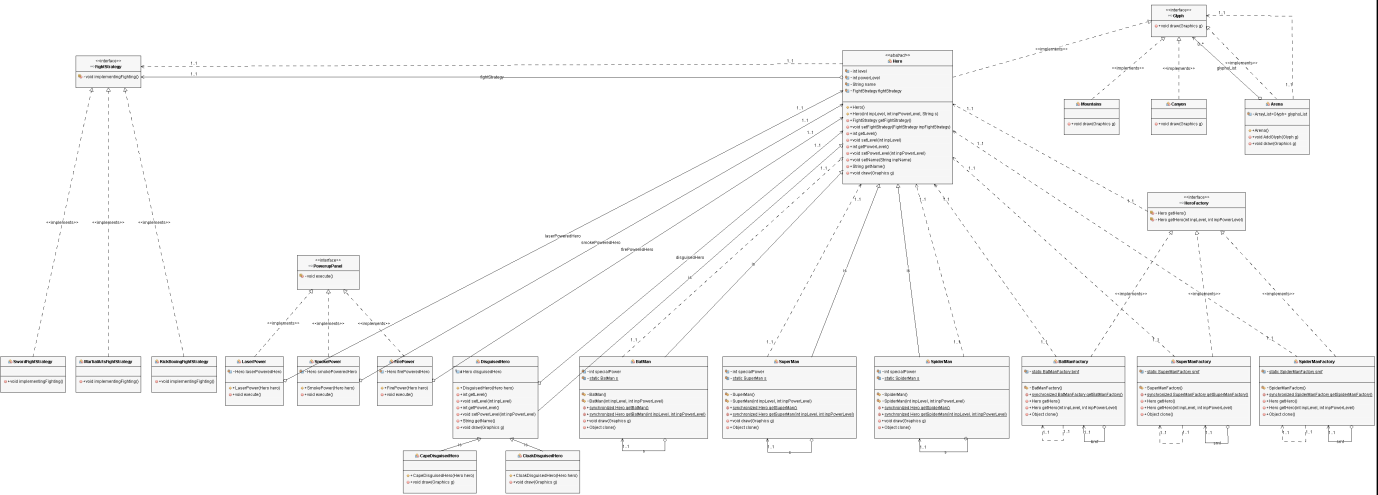


### Command Design Pattern

* This design pattern is used to execute the special power ups grabbed by the players during the fight.

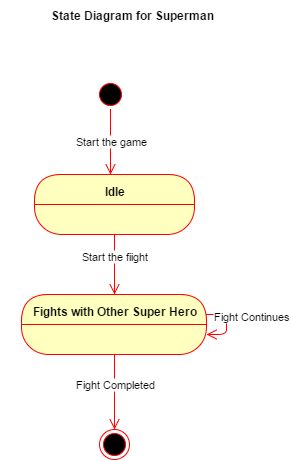


# Final Class Diagram

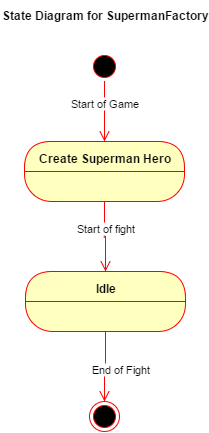


# State Diagrams

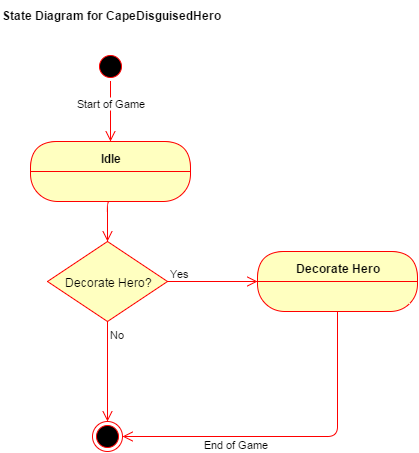
## Hero State Diagram



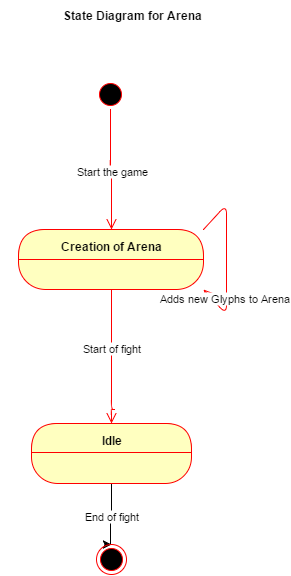
## HeroFactory State Diagram



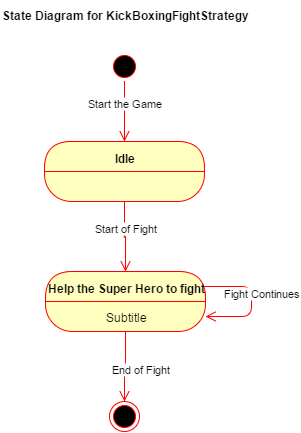
## DisguisedHero State Diagram



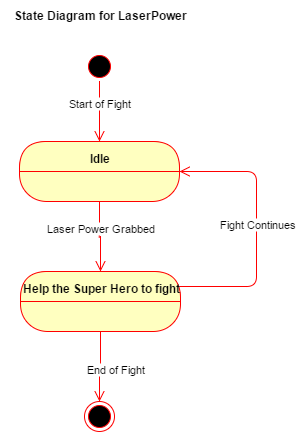
## Arena State Diagram



## FightStrategy State Diagram



## PowerUp State Diagram



# Code Snippets

## Singleton Design Pattern Code

public class BatMan extends Hero

{

private int specialPower;

private static BatMan s;

/\* Constructor for objects of class BatMan \*/

private BatMan(){

super("BatMan");

specialPower = 3;

}

/\* Method to create an instance of BatMan \*/

public static synchronized Hero getBatMan(){

if (s == null){

s = new BatMan();

}

return s;

}

/\* Method to override the clone() method \*/

public Object clone() throws CloneNotSupportedException{

throw new CloneNotSupportedException();

}

}

## Abstract Factory Design Pattern

public class BatManFactory implements HeroFactory{

private static BatManFactory bmf;

/\* Overriding the Default Constructor \*/

private BatManFactory(){

}

/\* Method to create an instance of BatManFactory \*/

public static synchronized BatManFactory getBatManFactory(){

if (bmf == null){

bmf = new BatManFactory();

}

return bmf;

}

/\* Overriding the getHero Method \*/

public Hero getHero() {

return BatMan.getBatMan();

}

/\* Method to override the clone() method \*/

public Object clone() throws CloneNotSupportedException{

throw new CloneNotSupportedException();

}

}

## Decorator Design Pattern

public class DisguisedHero extends Hero {

protected final Hero disguisedHero;

/\* Constructor for objects of class DisguisedHero \*/

public DisguisedHero(Hero hero) {

disguisedHero = hero;

}

/\* Overriding the draw method \*/

public void draw(Graphics g) {

disguisedHero.draw(g);

}

}

public class CapeDisguisedHero extends DisguisedHero

{

/\* Constructor for objects of class CapeDisguisedHero \*/

public CapeDisguisedHero(Hero hero){

super(hero);

}

/\* Overriding the draw method \*/

public void draw(Graphics g){

super.draw(g);

System.out.println(super.getName() + " is decorated with Cape");

}

}

## Composite Design Pattern

public class Arena implements Glyph{

private ArrayList<Glyph> glyphsList;

/\* Constructor for objects of class Arena \*/

public Arena() {

glyphsList = new ArrayList<>();

}

/\* Add a Glyph to the Arena \*/

public void AddGlyph(Glyph g) {

glyphsList.add(g);

}

/\* Overriding the abstract draw method \*/

public void draw(Graphics g){

System.out.println("This is the Arena");

for (Glyph glyph : glyphsList) {

glyph.draw(g);

}

}

}

## Strategy Design Pattern

public class SwordFightStrategy implements FightStrategy {

/\* Overriding the implementingFighting method \*/

public void implementingFighting () {

System.out.println("This object uses Swords to fight");

}

}

## Command Design Pattern

public class LaserPower implements PowerupPanel {

private Hero laserPoweredHero;

/\* Constructor for objects of class LaserPower \*/

public LaserPower(Hero hero){

laserPoweredHero = hero;

}

/\* Overriding the execute method \*/

public void execute(){

System.out.println("Using the Laser Power Up");

}

}