Dheeraj Prajwal Bojja Venkata
A20329437

CSP-585 Object Oriented Design Patterns
Final Exam Submission

Design Prototype

My game is inspired from a lot of other games.

Some of them include Mortal Kombat.

The design Prototype of Arena, Fighter Selection, gameplay , Power Bar and Experience bar and Summary after game.

Arena: Plates of Fire

Fighter Selection Page

Dheeraj Prajwal Bojja Venkata
A20329437

Game Play with Special Bar at bottom filled after a few Simple Kicks and Combo. Power Level at top denoting the power of each Player.

Player Win Game

War/Game XP calculated for each player at bottom

Result Showing the Score/Experience Earned and also showing the level.

Dheeraj Prajwal Bojja Venkata
A20329437

Game Rules

1. The game has 3 pairs of fighters pitted against each other Batman-Joker, Flash-Zoom and Scorpion-Subzero (from Mortal Kombat).
2. The game has a timer game timer and it is 60 seconds. In case of a draw, where both the fighters don't die and the power of the fighters are the same when the clock runs out, an extra timer of 30 seconds duration starts until the game finishes.
3. Game is in the Plates of arena and the fighters stay in the arena throughout the game.
4. Game is a one match knockout, winner of the first match is the game winner.
5. Game Fighters have some common basic moves like Kick, Punch, Move, Block, Jump and Defend.
6. These are also the states of the fighter along with Idle.
7. Game characters have their own special Moves which is unlocked after 4 basic moves or a Combo move (comprising of 4 basic moves).
8. Power level of each player is fixed and increases with Game wins
9. Experience is gained by player/fighter i.e. greater XP if won and smaller if lost.
10. There are 4 levels all players are initially in the level 0 and gain levels by gaining XP after every game.100-200 XP qualifies for level 1, 200-300 for level 2 and >300 for level 3.
11. User chooses his player and based on my fighter pairs algorithm the enemy/opponent is chosen automatically.
12. The fight happens only by checking whether both the fighters are of the same level.
13. The power of the player increases to a pre-determined algorithm which depends on number of moves made, game timer etc.
14. The players of Different levels have access to different Weapons like Bomb, Sword and Gun for levels 1, 2 and 3 respectively.
15. Most importantly there can be one and only one instance of each fighter.

Algorithms:

Algorithms Used for the game are as follows:

1. Fighter Selection
   Start Game
   Gaming menu
   Select a Game Character
   Singleton and abstract factory to create a fighter.
   Enemy selection happens through another algorithm.
   Game Timer Starts
   Fight begins
   End Game

2. Opponent Selection
   Start Game
   Gaming menu
   After the user selects a Game Character to fight, based on the selection his opponent is automatically chosen using Strategy pattern where Batman always fights Joker, Scorpion fights SubZero and Flash fights Zoom and Vice versa is also true.
   Game Timer Starts
   Fight begins
   End Game

3. Level Calculation

   Game Starts
   Set level=0 if not set.
   Get experience base value.
   This happens using Composite Pattern.
   Check if experience is between 0-100 then set the level as 1
   Similarly if it is between 101-200 then set level as 2
   And finally if it is between 201-300 then set level as 3.
   Game Timer Starts
   Fight begins
   End Game

4. Power Bar Calculation

   Game Starts
   Set power to 50 if not set.
   Get Power base value.
   Game Timer Starts
   Fight begins
   This happens using Composite Pattern.
   Moves are declared pre-defined values such as
      Punch = 3
      Kick = 5
      Special = 10
      Bomb = 8
      Sword = 10
      Gun = 12
   Reduce the power for every successful move and declare winner who still has power left (i.e. not depleted to 0).
   Game Ends

5. Experience bar calculation
   Game Starts
   Set power to 50 if not set.
   Get Power base value.
   Game Timer Starts
   Fight begins
   This happens using Composite Pattern.
   Moves are declared pre-defined values to gain XP such as
      Punch = 3
      Kick = 5
      Special = 10
      Bomb = 8
      Sword = 10
      Gun = 12
   And the winner gets additional bonus of 1 point for every second left in the game timer plus an additional 20 points for Winning.
   Game Ends
   Get The Experience Points of the fighter and update the points earned.

6. Weapons

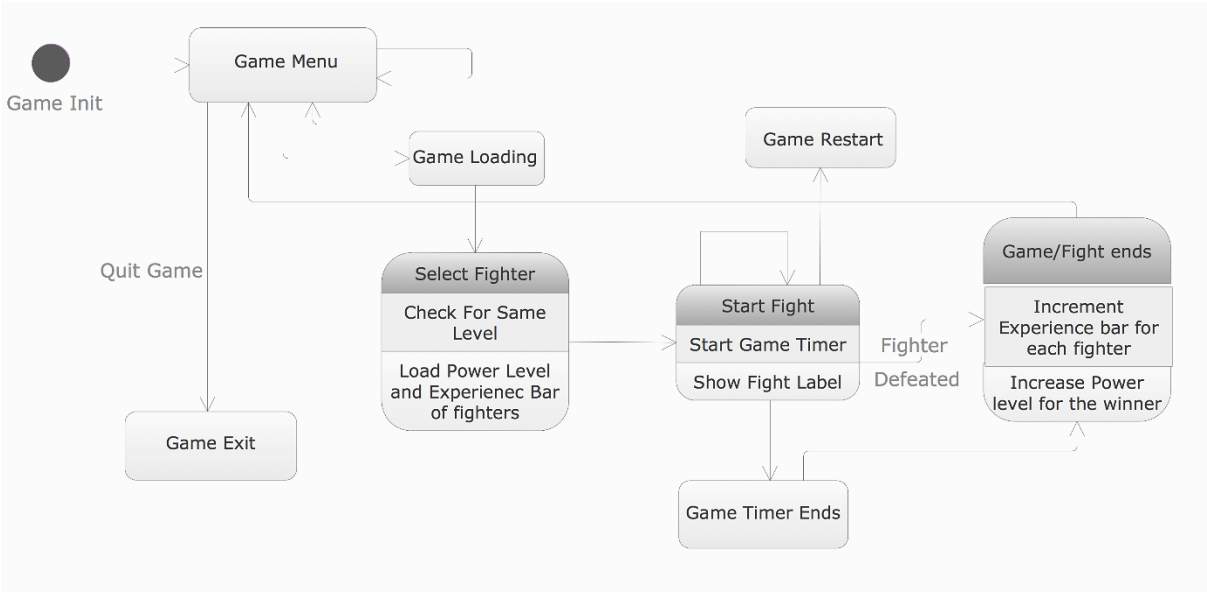
   Game Starts
   Game Timer Starts
   Check the Level of the players
   If Level=1 unlock and add weapon Bomb to both the players
   If Level=2 unlock and add weapon Sword to both the players
   If Level=3 unlock and add weapon Gun to both the players
   Fight begins
   Game Ends

<u>Design Patterns applied for designing this game are:</u>

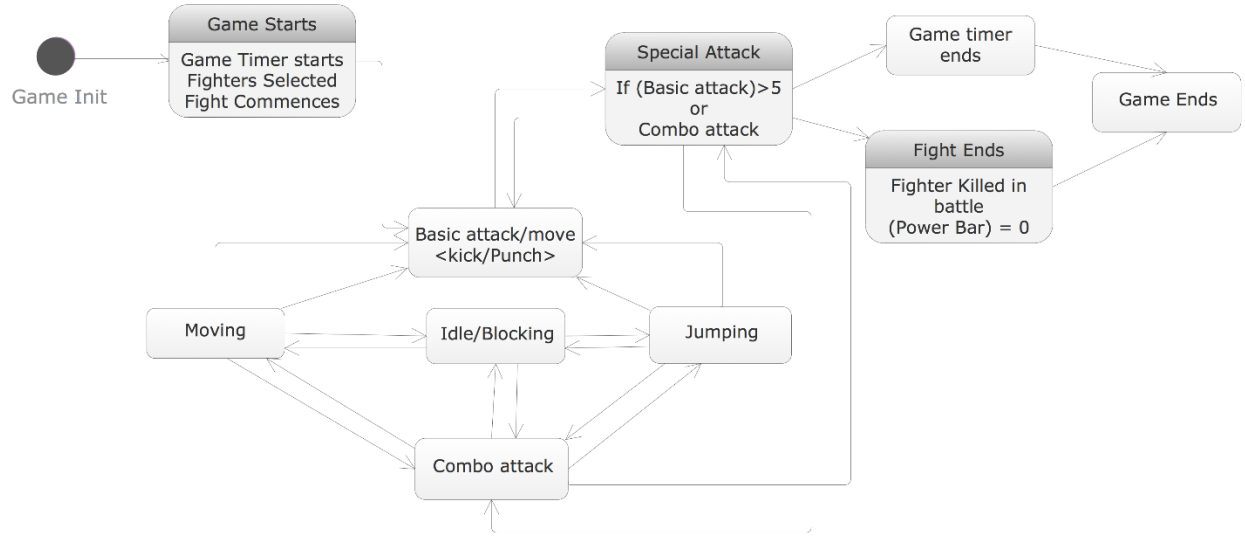
Type		
Creational	Singleton	Abstract Factory
Structural	Composite	Decorator
Behavioral	Strategy	Observer and State

Dheeraj Prajwal Bojja Venkata
A20329437

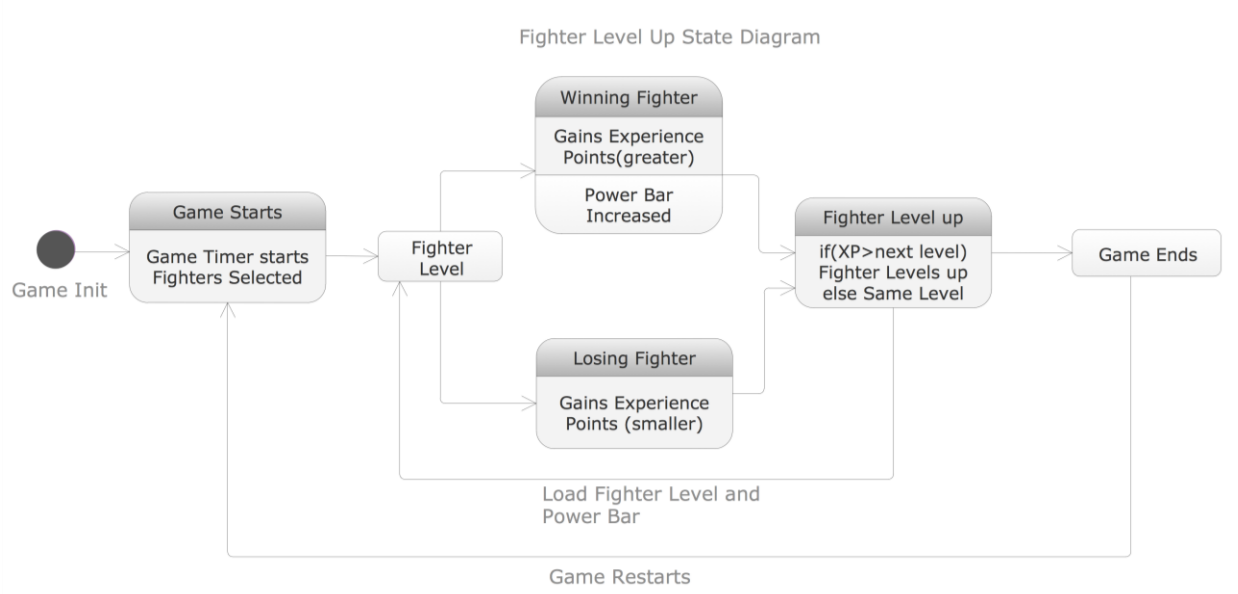
State Diagrams:



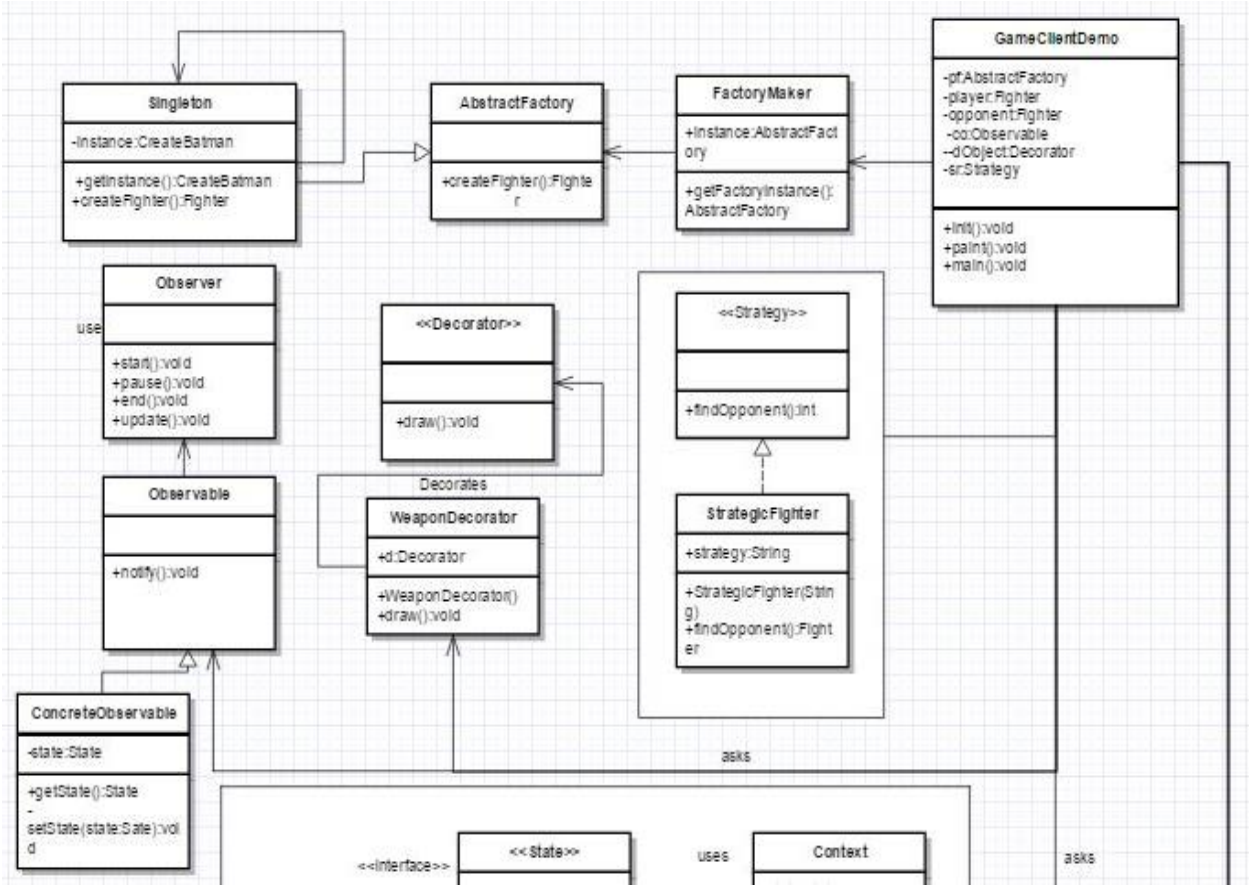
Fighter Attack State Diagram



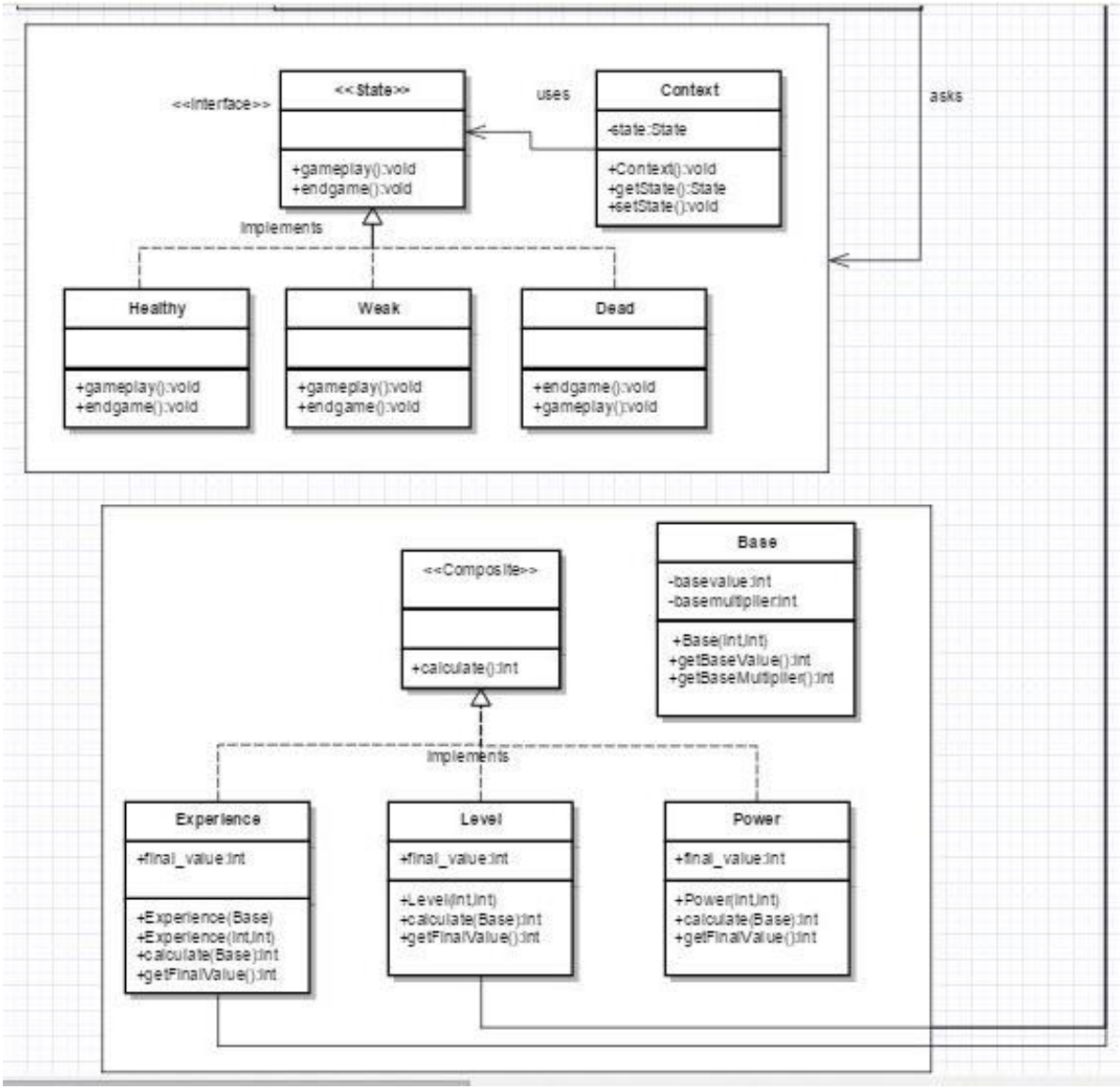
Dheeraj Prajwal Bojja Venkata
A20329437



Class Diagrams:

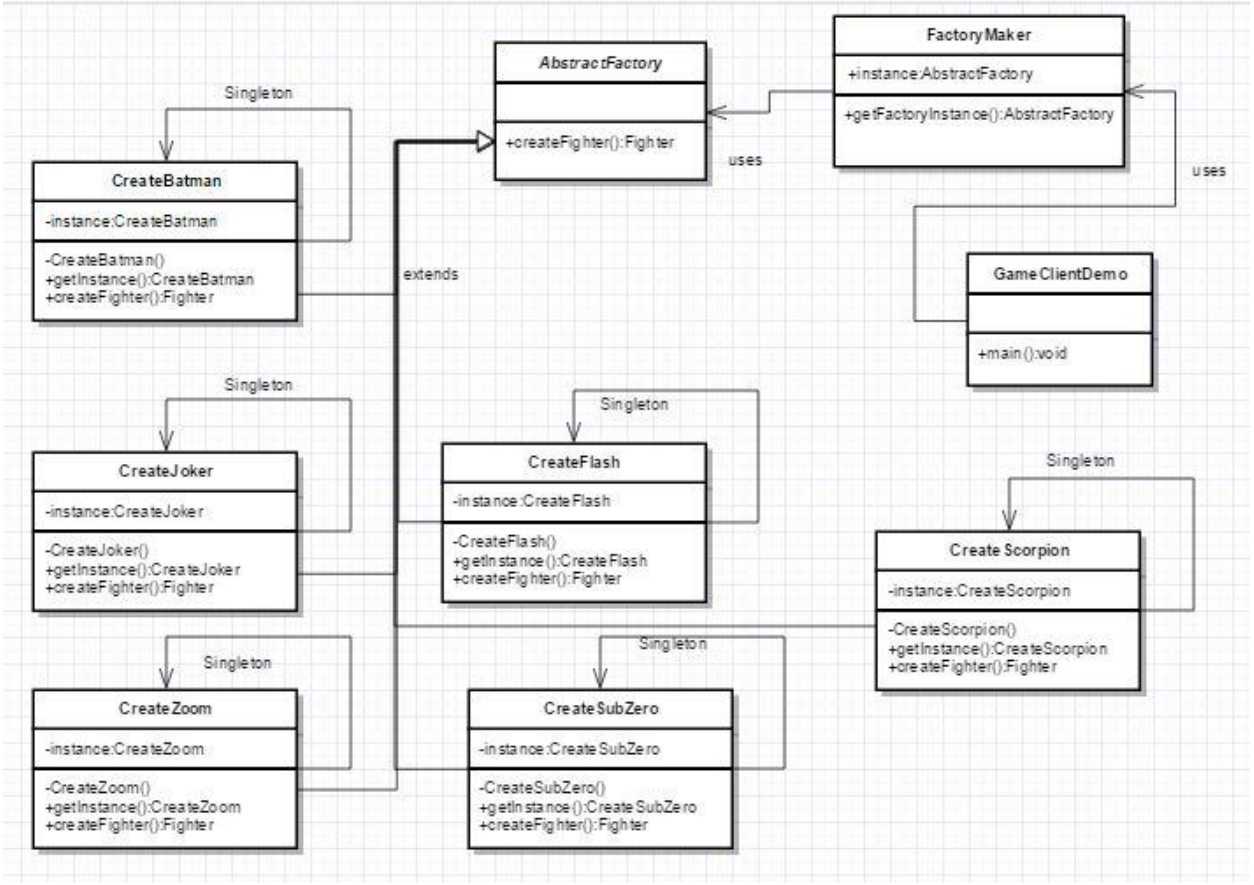


Dheeraj Prajwal Bojja Venkata
A20329437

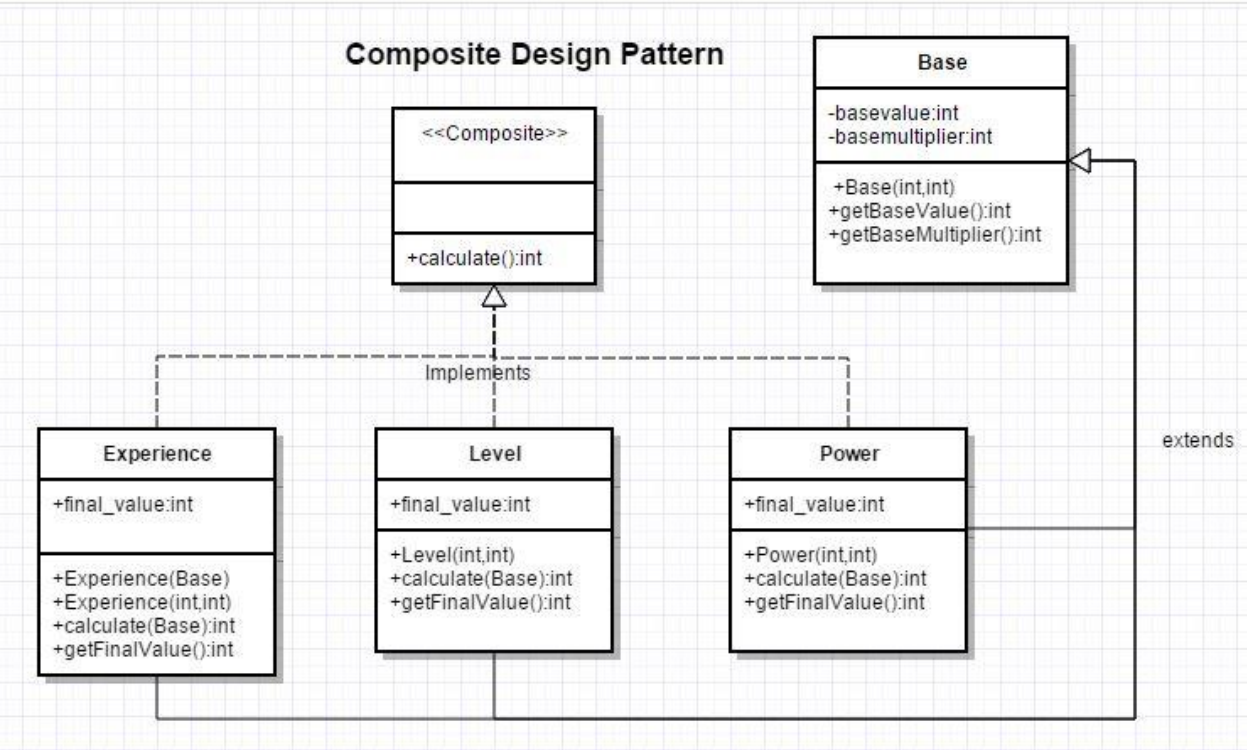


Dheeraj Prajwal Bojja Venkata
A20329437

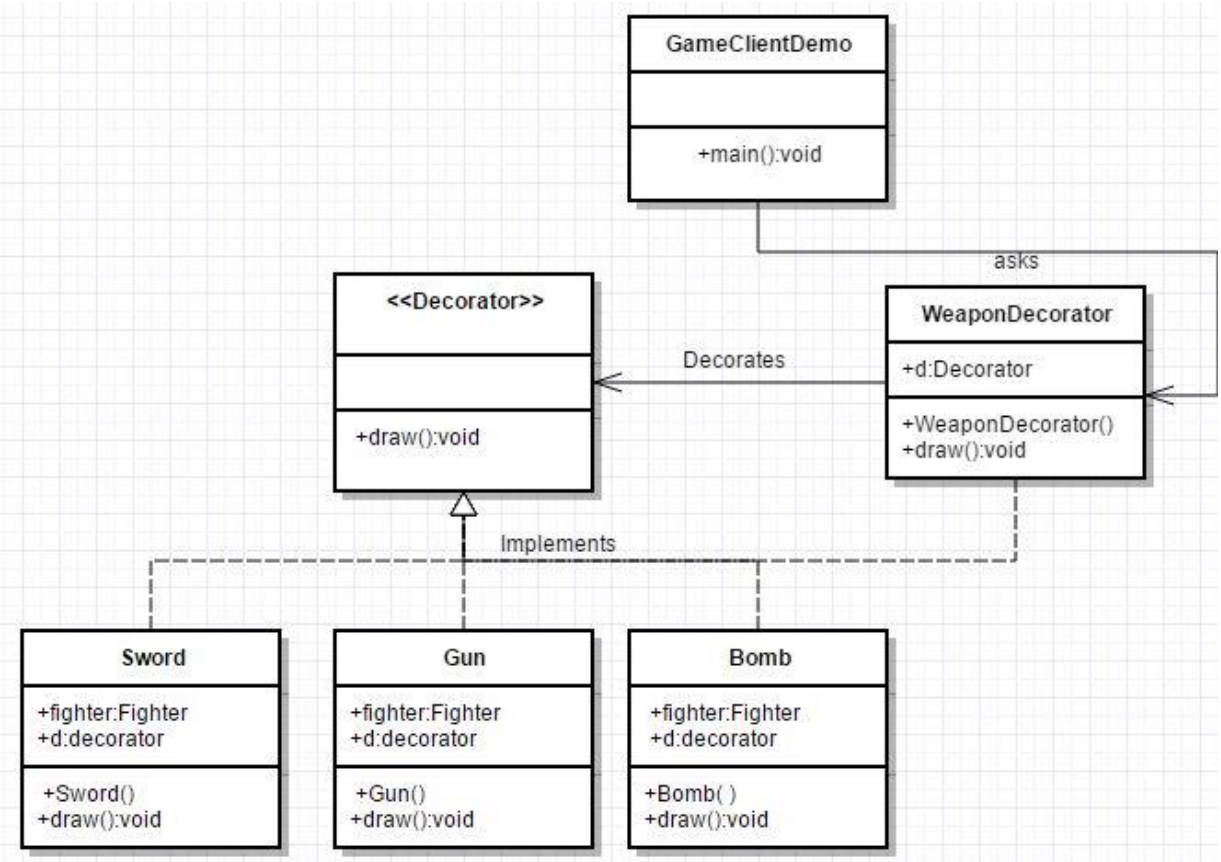
Singleton and Abstract Factory



Dheeraj Prajwal Bojja Venkata
A20329437

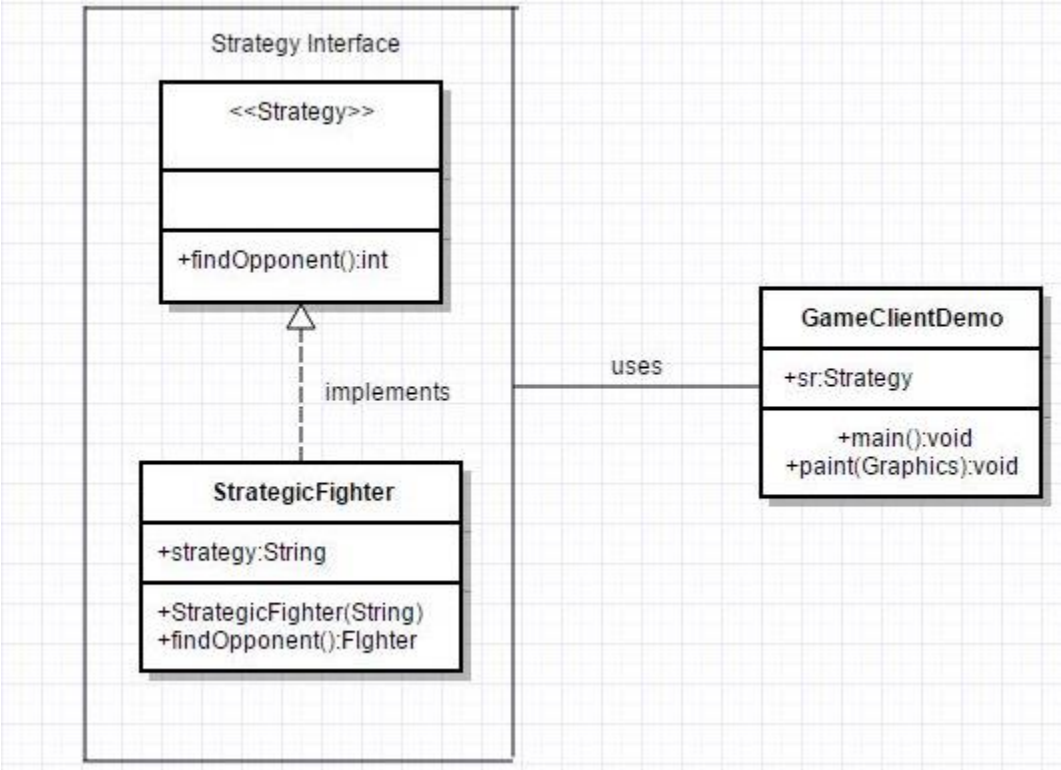


Decorator

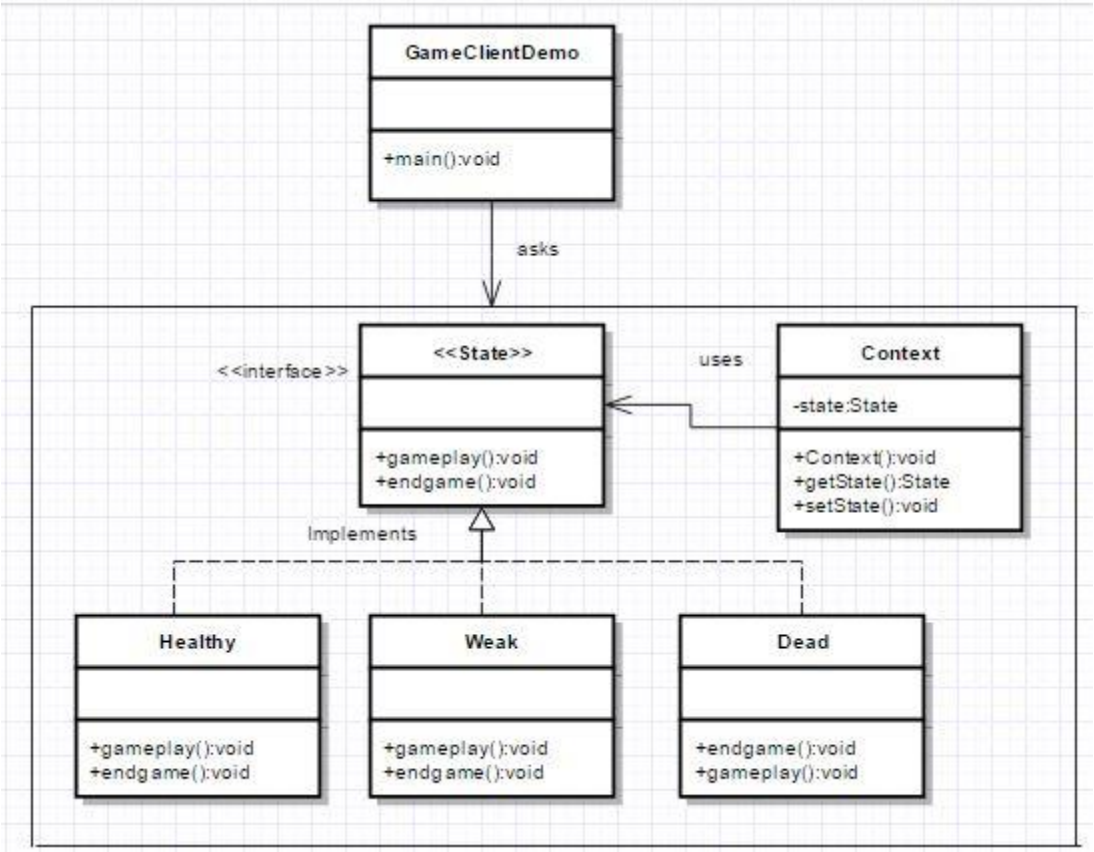


Dheeraj Prajwal Bojja Venkata
A20329437

Strategy

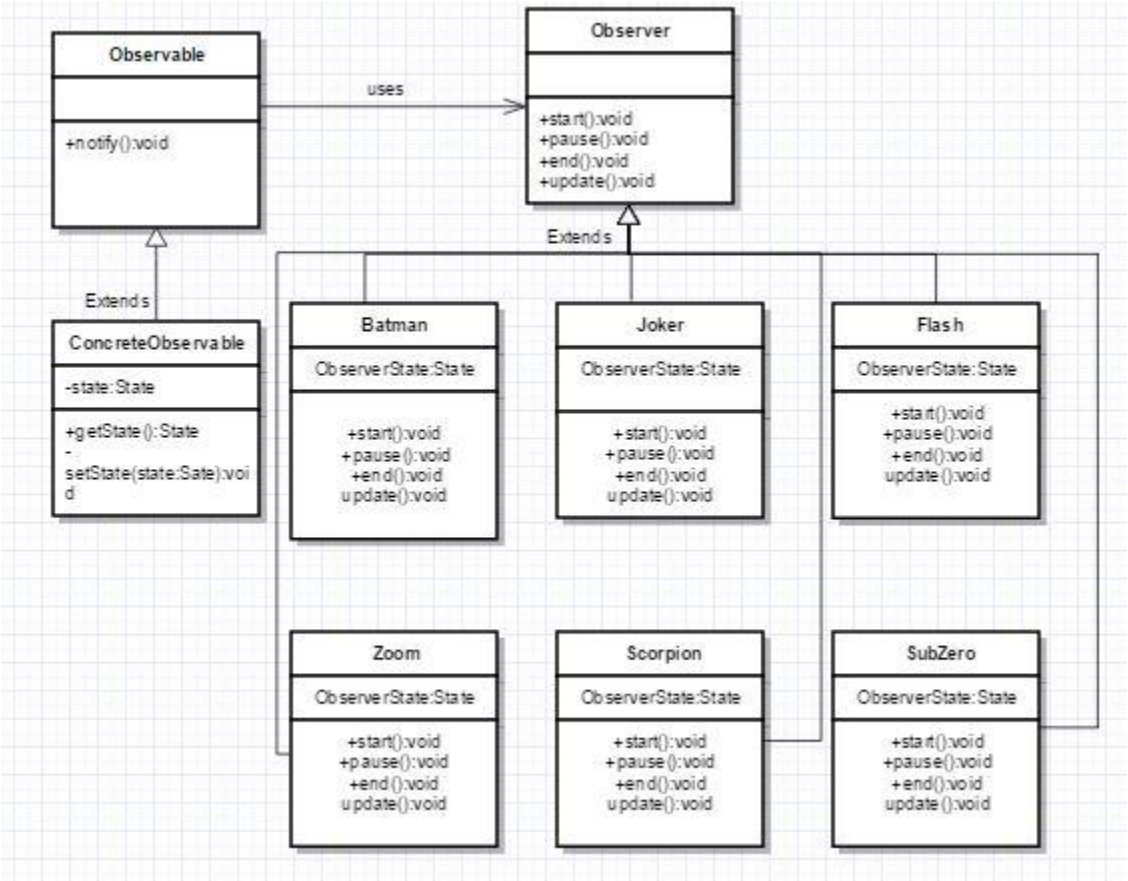


State



Dheeraj Prajwal Bojja Venkata
A20329437

Observer



Dheeraj Prajwal Bojja Venkata
A20329437

Code Snippets: Game Client Demo Class:

GameClientDemo.java

```
package com.iit.cs.oodp.exam;
import java.awt.Graphics;
import javax.swing.JApplet;

public class AbstractFactorySingletonClient extends JApplet{
    private AbstractFactory pf;
    private Fighter player;
    private Fighter opponent;
    private Strategy sr;
    Observable co;
    Decorator dobject;
      private static final long serialVersionUID = 1L;


    Context context = new Context();
    Healthy healthy = new Healthy();
    Weak weak = new Weak();
    Dead dead= new Dead();

    Base base;
         public void init( ) {
             resize(500,200);
             Arena.loadArena("file.gif");//code to load arena;
                pf=FactoryMaker.getFactoryInstance("Batman");
                player=pf.createFighter();
                if(player.getFighterLevel()==0)
                {
                    sr=new Strategy("batman");
                    opponent=sf.findOpponent();
                    Level l=new Level(0,1);
                }
                else if(player.getFighterLevel()==1)
                {
                    sr=new Strategy("batman");
                    opponent=sf.findOpponent();
                    Level l=new Level(1,1);
                    dobject=new WeaponDecorator(new Bomb());
```

```
            Level l=new Level(1,1);
            dobject=new WeaponDecorator(new Bomb());

        }else if(player.getFighterLevel()==2)
        {
            sr=new Strategy("batman");
            opponent=sf.findOpponent();
            Level l=new Level(2,1);
            dobject=new WeaponDecorator(new Sword());
        }else if(player.getFighterLevel()==3)
        {
            sr=new Strategy("batman");
            opponent=sf.findOpponent();
            Level l=new Level(3,1);
            dobject=new WeaponDecorator(new Gun());
        }
        //Code for other Players Selected
    }
    public void paint( Graphics g ) {
    for ( int i = 0; i < getWidth( )/7; i++ )
       {

        healthy.gameplay(context);
        System.out.println(context.getState().toString());
        co = new ConcreteObservable(healthy);
        co.notify();
        dobject.draw(g);
        player.draw(g);
        opponent.draw(g);
        //Code for fight to happen
        //The State sets to Weak in between
        weak.gameplay(context);
        System.out.println(context.getState().toString());
        co = new ConcreteObservable(weak);
        co.notify();
        //when the game ends the state sets to dead
        dead.gameplay(context);
        co = new ConcreteObservable(dead);
        co.notify();
        //Some code to Restart Game or Quit Game here
```

```
        co.notify();
        //Some code to Restart Game or Quit Game here
        if(base!=null)
        {
            Experience e=new Experience(base);
            Power p=new Power(base.getValue,base.getMultiplier);
            Level l= new Level(base.getValue,base.getMultiplier);
        }
        else
        {
            Experience e=new Experience(20,5);
            Power p=new Power(30,5);
            Level l= new Level(0,1);
        }

        try {
        Thread.sleep(50);
        }
        catch (InterruptedException e)
        {
        e.printStackTrace();
        }

      }
    System.exit(0);
  }
```

Singleton and Abstract Factory

FactoryMaker.java | AbstractFactory.java | CreateBatman.java | CreateFlash.java | CreateJo

```
package com.iit.cs.oodp.exam;

public class FactoryMaker {

private static AbstractFactory af=null;
static AbstractFactory getFactoryInstance(String fighterName){
                if(fighterName.equalsIgnoreCase("Batman")){
                af=CreateBatman.getInstance();
                }else if(fighterName.equalsIgnoreCase("Joker")){
                af=CreateJoker.getInstance();
                }else if(fighterName.equalsIgnoreCase("Flash")){
                af=CreateFlash.getInstance();
                }else if(fighterName.equalsIgnoreCase("Zoom")){
                af=CreateZoom.getInstance();
                }else if(fighterName.equalsIgnoreCase("Scorpion")){
                af=CreateScorpion.getInstance();
                }else if(fighterName.equalsIgnoreCase("SubZero")){
                af=CreateSubZero.getInstance();
                }return af;
        }
    }
}
```

Dheeraj Prajwal Bojja Venkata
A20329437

```java
package com.iit.cs.oodp.exam;

public abstract class AbstractFactory {
    abstract Fighter createFighter();
}
```

CreateBatman.java × CreateFlash.java × CreateJoker.java × Creat

```java
package com.iit.cs.oodp.exam;

import java.awt.Color;

public class CreateBatman extends AbstractFactory{
    private CreateBatman()
    {
    }
    public static CreateBatman getInstance()
    {
        return new CreateBatman();
    }
    public Fighter createFighter(){
        return new Fighter("Batman",50,108);
    }

}
```

CreateBatman.java × CreateFlash.java × CreateJoker

```java
package com.iit.cs.oodp.exam;

import java.awt.Color;

public class CreateJoker extends AbstractFactory{
    private CreateJoker()
    {
    }
    public static CreateJoker getInstance()
    {
        return new CreateJoker();
    }
    Fighter createFighter(){
        return new Fighter("Joker",50,108);
    }

}
```

Similarly For other Fighters

Fighter.java | Mortal.java | MortalFighter.java | Moves.java

```java
package com.iit.cs.oodp.exam;

import java.awt.*;

public class Fighter extends MortalFighter implements Moves{
    private static defaultLevel=0;
    Level fighter_level=new Level();
    public Fighter(String id,int x,int y) {
        super(id,x,y);
    }
    public void draw(Graphics g) {
        int X = getX( );
        int Y = getY( );
        String fighter=getID();
        //draw the fighter based on the character
        if(fighter.equalsIgnorecase("Batman"))
        {
            system.out.println("Batman Character Created");
        }else if(fighter.equals("Joker"))
        {
            system.out.println("Joker Character Created");
        }else if(fighter.equals("Flash"))
        {
            system.out.println("Flash Character Created");
        }else if(fighter.equals("Zoom"))
        {
            system.out.println("Zoom Character Created");
        }else if(fighter.equals("Scorpion"))
        {
            system.out.println("Scorpion Character Created");
        }else if(fighter.equals("SubZero"))
        {
            system.out.println("SubZero Character Created");
        }
        //Here goes the code to create those characters using fillPolygon or fillArc and/or setColor.
```

Dheeraj Prajwal Bojja Venkata
A20329437

```java
    }
    public int getFighterLevel()
    {
        return fighter_level.getFinalValue();

    }
    public void kick_power_calculator() {
        int Basic (int)(Math.random()*10);
        int Special =(int)(Math.random()*10);
        int Combo =(int)(Math.random()*7)
        int combo_power=Combo*basic_kick;
        int basic_power=Basic*basic_kick;
        int special_power=Special*special_kick;
}

package com.iit.cs.oodp;

public interface Moves{
    int basic_kick = 5;
    int special_kick = 7;
    void kick_power_calculator();
}
```

MortalFighter.java

```java
package com.iit.cs.oodp.exam;

import java.awt.Graphics;

    public abstract class Mortal {
        private int x; // x position
        private int y; // y position
        private String ID; // animal ID

        // constructor
        public Mortal() {
            ID = "";
        }
        // overloaded constructor
        public Mortal(String rID, int rX, int rY) {
            ID = rID;
            x = rX;
            y = rY;
        }
        public String getID() { return ID; }

        public int getX() {
            return x;
        }

        public int getY() {
            return y;
        }

        public void setX(int newX) {
            x = newX;
        }

        public void setY(int newY) {
            y = newY;
        }
        public abstract void draw( Graphics g );
    }
```

Dheeraj Prajwal Bojja Venkata
A20329437

Composite Pattern

Base.java | Composite.ja

```java
package com.iit.cs.oodp.exam;

public class Base
{
    private int basevalue;
    private int basemultiplier;
    public Base(int bv,int bm)
    {
        basevalue=bv;
        basemultiplier=bm;
    }
    public int getBaseValue()
    {
        return basevalue;
    }
    public int getBaseMultiplier()
    {
        return basemultiplier;
    }
}

package com.iit.cs.oodp.exam;

public Interface Composite{
    public int calculate();
}
```

Power.java | Experience.java | Level.ja

```java
package com.iit.cs.oodp.exam;

public class Power extends Base implements Composite{
    public int final_value;
    public Power(int value,int multiplier)
    {
        super(value,multiplier);

    }
    public int calculate(Base b)
    {
        int value=b.getBaseValue();
        int multiplier=b.getBaseMultiplier();
        final_value=value+(value*(multiplier/100));
        return final_value;
    }
    public int getFinalValue()
    {
        return final_value;
    }
}
```

Experience.java | Level.java

```java
package com.iit.cs.oodp.exam;

public class Experience extends Base implements Composite{
    public int final_value;
    public Experience(Base ba)
    {
        super(ba.basevalue,ba.basemultiplier);

    }
    public Experience(int value,int multiplier)
    {
        //constructor overloading
        super(value,multiplier);

    }
    public int calculate(Base b)
    {
        int value=b.getBaseValue();
        int multiplier=b.getBaseMultiplier();
        if(game.equals("Win"))
        {
            final_value=value+(value*((multiplier+10)/100));
        }
        else if(game.equals("Lose"))
        {
            final_value=value+(value*((multiplier+3)/100));
        }
        return final_value;
    }
    public int getFinalValue()
    {
        return final_value;
    }
}
```

Dheeraj Prajwal Bojja Venkata
A20329437

Level.java

```java
package com.iit.cs.oodp.exam;

public class Level extends Base implements Composite{
    public int final_value;
    public Level(int value,int multiplier)
    {
        super(value,multiplier);

    }
    public int calculate(Base b)
    {
        int value=b.getBaseValue();
        int multiplier=b.getBaseMultiplier();
        Base XP=new Experience(b);
        int finalXP=XP.getFinalValue();
        if(XP>100 && XP < 200)
        {   //level 1 upgrade from level 0
            final_value=value+multiplier;
        }
        else if(XP>200 && XP < 300)
        {
            //level 2 upgrade from level 1
            final_value=value+multiplier;
        }
        else if(XP>300)
        {
            //level 3 upgrade from level 2 max level
            final_value=value+multiplier;
        }
        else
        {
            //level 3 max so keep it that way.
            final_value=0;
        }
        return final_value;
    }
    public int getFinalValue()
    {
        return final_value;
    }
```

Dheeraj Prajwal Bojja Venkata
A20329437

Decorator Design Pattern

Decorator.java | WeaponDecorator.java | Bomb.java | Sw

```java
package com.iit.cs.oodp;

import java.awt.Graphics;
//Class that is required to be extended in-order to Create the Decorators
public interface Decorator {
    public void draw(Graphics g);
}

package com.iit.cs.oodp;
import java.awt.Graphics;
//MyDraw to Extend the Decorator
public abstract class WeaponDecorator implements Decorator {
    protected Decorator d;

   public WeaponDecorator(Decorator d){
      this.d = d;
   }
   public abstract void draw(Graphics g);

   }


```

Bomb.java | Sword.java | Gun.jav

```java
package com.iit.cs.oodp;
import java.awt.Color;
import java.awt.Graphics;

public class Sword implements Decorator{
    protected Decorator d;
    protected Fighter fighter;
    public Sword(Decorator dObject,Fighter player)
    {
        d = dobject;
        fighter = player;
    }
    public void draw(Graphics g)
    {    //Draw The Sword Here
         d.draw(g);
    }
}
```

Dheeraj Prajwal Bojja Venkata
A20329437

Bomb.java | Sword.java | Gun

```java
package com.iit.cs.oodp;
import java.awt.Color;
import java.awt.Graphics;

public class Bomb implements Decorator{
    Decorator d;
    Fighter fighter;
    public Bomb(Decorator dObject,Fighter player)
    {
        dobject = d;
        fighter = player;
    }
    public void draw(Graphics g)
    {     //Draw The Bomb Here
        d.draw(g);
    }
}
```

Gun.java

```java
package com.iit.cs.oodp;
import java.awt.Color;
import java.awt.Graphics;

public class Gun implements Decorator{
    Decorator d;
    Fighter fighter;
    public Gun(Decorator dObject,Fighter player)
    {
        d = dobject;
        fighter = player;
    }
    public void draw(Graphics g)
    { //Draw The Gun Here
      d.draw(g);
    }
}
```

Dheeraj Prajwal Bojja Venkata
A20329437

Strategy Design Pattern:

StrategicFighter.java

```java

package com.iit.cs.oodp.exam
public class StrategicFighter implements Strategy {
  public String strategy;

   public StrategicFighter(String fighter){
      this.strategy = fighter;
   }
   public Fighter findOpponent(){
      if(strategy.equalsIgnoreCase("Batman"))
      {
        return CreateJoker.getInstance();
      }
      else if(strategy.equalsIgnoreCase("joker"))
      {
        return CreateBatman.getInstance();
      }
      else if(strategy.equalsIgnoreCase("Flash"))
      {
        return CreateZoom.getInstance();
      }
      else if(strategy.equalsIgnoreCase("Zoom"))
      {
        return CreateFlash.getInstance();
      }
      else if(strategy.equalsIgnoreCase("Scorpion"))
      {
        return CreateSubZero.getInstance();
      }
      else if(strategy.equalsIgnoreCase("SubZero"))
      {
        return CreateScorpion.getInstance();
      }
   }
}
package com.iit.cs.oodp.exam
public interface Strategy {
   public int findOpponent();
}
```

Dheeraj Prajwal Bojja Venkata
A20329437

Observer Design Pattern

ConcreteObservable.java | Observable.java

```java
package com.iit.cs.oodp.exam
class ConcreteObservable extends Observable{
State st;
public ConcreteObservable(State s)
{
    super(s);
    st=s;
}
public void notify()
{
this.st=notify();
}
}
package com.iit.cs.oodp.exam
public class Observable{
     Context c=new Context();
     State s=c.getState();
     public Observable(State s)
     {
      if(s!=(null))
      this.s=s;
     }
     public void notify()
     {
          state=c.setState(new Healthy());
          //Code to notify all fighters
     }
}
```

Dheeraj Prajwal Bojja Venkata
A20329437

State Design Pattern:

Dead.java | Healthy.java | Weak.java

```java
public class Dead implements State {
    public void endgame(Context context) {
        System.out.println("Player Dead");
        //Player Died and Code to restart or Quit the game to Game menu
        context.setState(this);
    }
    public String toString(){
        return "Dead";
    }
}
public class Healthy implements State {
    public void gameplay(Context context) {
        System.out.println("Player is Healthy and ready to Fight");
        context.setState(this);
    }
    public String toString(){
        return "Healthy";
    }
}
public class Weak implements State {
    public void weak(Context context) {
        System.out.println("Player is weak and ready to die");
        //code to defend or attack faster
        context.setState(this);
    }
    public String toString(){
        return "Weak";
    }
}
```

Context.java | State.java

```java
package com.iit.cs.oodp;
public class Context {
    private State state;
    public Context(){
        state = null;
    }
    public void setState(State state){
        this.state = state;
    }
    public State getState(){
        return state;
    }
}
package com.iit.cs.oodp;
public interface State {
    public void gameplay(Context context);
}
```