CSE 3902 Midterm Sample Midterm (longer than will be given)

1. Give an example where the Singleton design pattern would be useful.

2. Label the following design patterns as to whether they are creational design patterns, structural design patterns or behavioral design patterns:
   (a) Abstract Factory
   (b) Builder
   (c) Proxy
   (d) Singleton
   (e) Observer
   (f) Decorator

3. What are the common and different characteristics of the *State* and the *Strategy* design patterns?
4. Design Pattern recognition (10 points)

Consider the following UML diagram.

Tasks:
   (a) First, identify the pattern in the diagram.
   (b) Second, identify the participants in the diagram and indicate their responsibilities.

5. Which design pattern you would you use to control the creation of an object based on an established interface, while allowing the concrete implementation to determine the subclass to construct.

Please choose only one answer:

- Singleton design pattern
- Builder Factory design pattern
- Prototype factory design pattern
- Factory method design pattern

6. Which design pattern you would you use to limit the class instantiation to one object?

Please choose only one answer:

- Factory Method Design Pattern
- Builder design pattern
- Prototype design pattern
- Singleton design pattern

7. When would you use the builder design pattern? (choose all that apply)

Please choose all the answers that apply:

a) to abstract steps of construction of complex objects
b) to build different representations of complex objects based on the concrete implementations of
c) construction procedure
d) to establish an interface for creating an object, but let the concrete implementations decide which
e) subclass to instantiate
f) to encapsulate a family of individual factories that have a common theme

8. Which design pattern is applied in the code snippet below?
Please choose only one answer:

- PrintSpooler design pattern
- Spooler design pattern
- Singleton design pattern
- Factory design pattern
- Abstract Singleton design pattern

```
public class PrintSpooler {
        private static final PrintSpooler instance = new PrintSpooler();
        private PrintSpooler() {}
        public static PrintSpooler getInstance() {
                return instance;
        }
}
```

9. What are the consequences of applying the Prototype pattern?
Please choose all the answers that apply:

- each concrete prototype class must implement the clone method
- it makes it easier for a certain family of objects to work together
- it enables the client code to register an new concrete prototype instance at run time
- it reduces of the class hierarchy as compared to the other factory design patterns

10. This is a problem about design patterns. For each part, write down the name of the design pattern or principle that would be most useful for addressing the situation described.
    a. You are building a system that relies on a complex algorithm, and that algorithm may be changed often due to marketing pressures. What pattern would best support this?
    b. You are building a system in which you want to separate the domain logic layer from the user interface layer. In particular, you want to allow changes in the state of the domain logic layer objects to be reflected in changes to one or more objects in the user interface layer. The domain logic layer shouldn’t know about the objects in the user interface layer, or even how many there are. What pattern would best support this?

11. Examine the code below and answer the following:
    c. What is the Design Pattern?
    d. What would be a better name for the class?

```
public static class Foo
{
    /// <summary>
    /// Creates a concrete <typeparamref name="IPriorityCollection{T}"/> instance.
    /// </summary>
    /// <typeparam name="T">Specifies the type of elements in the collection.</typeparam>
    /// <param name="queueType">Provides a string-based descriptor of the desired
    /// concrete implementation.</param>
    /// <param name="capacity">The initial number of elements that the
    /// <typeparamref name="IPriorityCollection{T}"/> can contain.</param>
    /// <returns>An instance of an <typeparamref name="IPriorityCollection{T}"/>.
    /// </returns>
    public static IPriorityCollection<T> createQueue<T>(string queueType, int capacity)
    {
        if (queueType == "RingBuffer")
            return new RingBuffer<T>(capacity);
        else if (queueType == "Heap")
            return new HeapAdaptor<T>(capacity);
        else if (queueType == "Queue")
            return new QueueAdaptor<T>(capacity);
        else if (queueType == "Stack")
            return new StackAdaptor<T>(capacity);
        else if (queueType == "CurrentState")
            return new CurrentStateBuffer<T>();

        return new HeapAdaptor<T>(capacity);
    }
}
```

12. Provide at least 3 Code Smells for the following code snippet from a single class. Justify your choices.

```
protected override void CollideWithGoomba(MovingEntity target)
{
    //check collision from above
    float difference = CollisionCalculator.CollideFromAbove(self, target);
    if (difference >= 0.0f)
    {
        //set it to fall as close to the target as possible
        difference -= .001f;
        self.CurrentVelocityY = -difference;
        self.OnGround = true;
    }

    //if a goomba collides with another goomba, it changes direction
    //check collision from the left
    float currentDistance = CollisionCalculator.CollideFromLeft(self, target);
    if (currentDistance >= 0.0f)
    {
        self.CurrentActions.FlipDirection();
        target.CurrentActions.FlipDirection();

    }
    else
    {
        //check from the right
        currentDistance = CollisionCalculator.CollideFromRight(self, target);
        if (currentDistance >= 0.0f)
        {
            target.GetHurt();
        }
    }
}

protected override void CollideWithStaticEntity(Entity target)
{
    /*
     * Checks for a collision in both directions
     * If there is a collision, sets the velocity so that it
     * gets as close as possible without overlapping
     * (well, not quite as close as possible because it causes
     * issues, probably because of float. It'll get really close though)
     */

    //check collision from above
    float difference = CollisionCalculator.CollideFromAbove(self, target);
    if (difference >= 0.0f)
    {
        //set it to fall as close to the target as possible
        difference -= .001f;
        self.CurrentVelocityY = -difference;
        self.OnGround = true;
    }
    else
    {
        //check collision from below
        difference = CollisionCalculator.CollideFromBelow(self, target);
        if (difference >= 0.0f)
        {
            //set it to get as close to the target as possible
            difference -= .001f;
            self.CurrentVelocityY = difference;
        }
    }
```

```
        //check collision from the left
        difference = CollisionCalculator.CollideFromLeft(self, target);
        if (difference >= 0.0f)
        {
            //set it to get as close to the target as possible
            difference -= .001f;
            self.CurrentVelocityX = difference;
            //after bumping into the wall, change direction
            self.CurrentActions.FlipDirection();
        }
        else
        {
            //check from the right
            difference = CollisionCalculator.CollideFromRight(self, target);
            if (difference >= 0.0f)
            {
                difference -= .001f;
                self.CurrentVelocityX = -difference;
                //after bumping into the wall, change direction
                self.CurrentActions.FlipDirection();
            }
        }
    }
```

13. Rework the following to use C# iterators and the yield return syntax.

```
        /// <summary>
        /// Returns a set of all level game objects within the
        /// bounds of the camera.
        /// </summary>
        /// <returns>Set of game objects within bounds of camera.</returns>
        private IList<IGameElement> getLevelObjectsInCamera()
        {
            IList<IGameElement> levelObjects = level.gameElements;
            IList<IGameElement> targetObjects = new List<IGameElement>();
            foreach (IGameElement gameObject in levelObjects)
            {
                // Add all game objects within the bounds of the camera to the targetObjects set.
                if (gameObject.Position.x <= this.borderRight && (gameObject.Position.x + 16) >=
this.borderLeft)
                {
                    if (gameObject.Position.y <= this.borderTop && gameObject.Position.y >=
this.borderBottom)
                    {
                        targetObjects.Add(gameObject);
                    }
                }
            }
            return targetObjects;
        }
```

1. Provide the major code smell with this class:

```
    internal class MushroomPhysics
    {
        internal Vector2 Position { get; private set; }
        private Vector2 Speed;
        internal Vector2 Velocity { get; set; }
        internal bool MovingRight { get; set; }

        public MushroomPhysics()
        {
            this.Position = new Vector2(0, 0);
            this.MovingRight = true;
            this.Speed = new Vector2(0, -3);
        }

        /// <summary>
        /// The Mushroom can move left.
        /// </summary>
        internal void MoveLeft()
        {
            this.Position.x = -2;
        }

        /// <summary>
        /// The Mushroom can move right.
        /// </summary>
        internal void MoveRight()
        {
            this.Position.x = 2;
        }

        /// <summary>
        /// Grounds the subject. This action affects the vertical
        /// acceleration calculations.
        /// </summary>
        internal void Ground()
        {
            this.Speed.y = 0;
        }

        public void Update()
        {
            if (this.MovingRight)
            {
                MoveRight();
            }
            else
            {
                MoveLeft();
            }
            this.Position.y += this.Speed.y;
        }
    }
}
```

2. Give a Definition of Cohesion and the desired property in software.
3. Explain whether (and why) the following code has high or low Cohesion.

```
public override void Update()
{
    HorizontalCollision();
    VerticalCollision();

    if (currentSprite == walkAnimationLeft)
    {
        walkAnimationLeft.AdvanceOneFrame();
    }
    else if (currentSprite == walkAnimationRight)
    {
        walkAnimationRight.AdvanceOneFrame();
    }

    base.Update();
}
```

1. Give at least 3 Code Smells for the following code:

```
public void Update(Movement currentMovement)
{
    if (currentMovement == Movement.LEFT)
    {
        AnimatedMario.AdvanceOneFrame();
        if (AnimatedMario.CurrentFrame < 14 || AnimatedMario.CurrentFrame > 
AnimatedMario.NumberOfFrames)
        {
            AnimatedMario.SetFrame(14);
        }
    }
    else if (currentMovement == Movement.RIGHT)
    {
        AnimatedMario.AdvanceOneFrame();
        if (AnimatedMario.CurrentFrame > 2)
        {
            AnimatedMario.SetFrame(0);
        }
    }
    else if (currentMovement == Movement.RUNLEFT)
    {
        AnimatedMario.AdvanceOneFrame();
        AnimatedMario.AdvanceOneFrame();
        if (AnimatedMario.CurrentFrame < 14 || AnimatedMario.CurrentFrame > 
AnimatedMario.NumberOfFrames)
        {
            AnimatedMario.SetFrame(14);
        }
    }
    else if (currentMovement == Movement.RUNRIGHT)
    {
        AnimatedMario.AdvanceOneFrame();
        AnimatedMario.AdvanceOneFrame();
        if (AnimatedMario.CurrentFrame >= 6)
        {
            AnimatedMario.SetFrame(0);
        }
    }
    else if (AnimatedMario.Velocity.y > 0)
    {
        if (AnimatedMario.Velocity.x >= 0)
        {
            AnimatedMario.SetFrame(6);
        }
        else
        {
            AnimatedMario.SetFrame(12);
        }
    }
    else if (currentMovement == Movement.NONE)
    {
        if (AnimatedMario.Velocity.x > 0)
        {
            AnimatedMario.SetFrame(0);
        }
        else if(AnimatedMario.Velocity.x < 0)
        {
            AnimatedMario.SetFrame(14);
        }
    }
    MarioPhysics.Gravity(AnimatedMario);
    controller.Move(AnimatedMario, currentMovement, wS);

}
```

2. Discuss the Cohesion and Coupling and give at least 4 Code Smells for the following:

```
/// <summary>
/// This initially draws the playable world to the window.
/// </summary>
public void Initialize()
{
    Vector2 position = new OhioState.Graphics.Math.Vector2(0, 0);
             string levelWidth ="",levelHeight ="";


    //Will execute until an exception is thrown
    try
    {
        using (StreamReader streamReader = new StreamReader(@"../../Documents/level1-1.csv"))
        {
            Char EntityCodeFromStream = (Char)streamReader.Read();//skip W
            //Console.WriteLine(EntityCodeFromStream);
            EntityCodeFromStream = (char)streamReader.Read();//skip ,
            //Console.WriteLine(EntityCodeFromStream);
                           //Read in width
                           EntityCodeFromStream = (char)streamReader.Read();
            Console.WriteLine(EntityCodeFromStream);
                           while (EntityCodeFromStream != ',')
            {
                levelWidth += EntityCodeFromStream;
                EntityCodeFromStream = (char)streamReader.Read();
                //Console.WriteLine(EntityCodeFromStream);
            }
            coordinateFrame.xAxis.x = Convert.ToInt32(levelWidth);

            EntityCodeFromStream = (char)streamReader.Read();//skip H
            //Console.WriteLine(EntityCodeFromStream);
            EntityCodeFromStream = (char)streamReader.Read();//skip ,
            //Console.WriteLine(EntityCodeFromStream);
                           //Read in height
                           EntityCodeFromStream = (char)streamReader.Read();
            //Console.WriteLine(EntityCodeFromStream);
                           while (EntityCodeFromStream != ',')
            {
                levelHeight += EntityCodeFromStream;
                EntityCodeFromStream = (char)streamReader.Read();
                //Console.WriteLine(EntityCodeFromStream);
            }
            coordinateFrame.yAxis.y = Convert.ToInt32(levelHeight);
            position.y = coordinateFrame.yAxis.y-1;
                           while (EntityCodeFromStream != ';') //skip to first line of level
            {
                EntityCodeFromStream = (char)streamReader.Read();
                //Console.WriteLine(EntityCodeFromStream);
            }

            //Print Tiles in Background

            float horizontalPosition = 0;

            while (horizontalPosition < coordinateFrame.xAxis.x)
            {
                IWorldObject backgroundTile = new Background(new Vector2(horizontalPosition,
0.7f), worldObjectCollection);
                worldObjectCollection.AddToWorld(backgroundTile);
                horizontalPosition += backgroundTile.coordinateFrame.xAxis.x;
            }
```

```
                    EntityCodeFromStream = (char)streamReader.Read();
                    while ((Char)EntityCodeFromStream != '!')
                    {
                        //Console.WriteLine(EntityCodeFromStream);
                        switch (EntityCodeFromStream)
                        {
                            case 'I':
                                IWorldObject blockStairs = new BlockStairs(new Vector2(position.x, 
position.y), worldObjectCollection);
                                worldObjectCollection.AddToWorld(blockStairs);
                                break;
                            case 'F':
                                IWorldObject block = new Block(new Vector2(position.x, position.y), 
worldObjectCollection);
                                worldObjectCollection.AddToWorld(block);
                                break;

                            case 'R':
                                IWorldObject brick = new Brick(new Vector2(position.x, position.y), 
worldObjectCollection);
                                worldObjectCollection.AddToWorld(brick);
                                break;

                            case 'C':
                                IWorldObject questionBlockCoin = new QuestionBlockCoin(new 
Vector2(position.x, position.y), worldObjectCollection);
                                worldObjectCollection.AddToWorld(questionBlockCoin);
                                break;

                            case '?':
                                IWorldObject questionBlockItem = new QuestionBlockItem(new 
Vector2(position.x, position.y), worldObjectCollection);
                                worldObjectCollection.AddToWorld(questionBlockItem);
                                break;

                            case '$':
                                IWorldObject questionBlockStar = new QuestionBlockStar(new 
Vector2(position.x, position.y), worldObjectCollection);
                                worldObjectCollection.AddToWorld(questionBlockStar);
                                break;

                            case 'P':
                                ISpriteFlyweight pipeSpriteLeft = 
SpriteEngine.Instance.SpriteFactory.CreateSprite("greenpipeleft");
                                IWorldObject pipeLeft = new Pipe(new Vector2(position.x, position.y), 
worldObjectCollection, pipeSpriteLeft);
                                worldObjectCollection.AddToWorld(pipeLeft);
                                break;
                            case '[':
                                ISpriteFlyweight pipeSpriteright = 
SpriteEngine.Instance.SpriteFactory.CreateSprite("greenpiperight");
                                IWorldObject pipeRight = new Pipe(new Vector2(position.x, position.y), 
worldObjectCollection, pipeSpriteright);
                                worldObjectCollection.AddToWorld(pipeRight);
                                break;
                            case 'T':

                                IWorldObject pipeTopLeft = new Pipe(new Vector2(position.x, 
position.y), worldObjectCollection,SpriteEngine.Instance.SpriteFactory.CreateSprite("greentopleft"));
                                worldObjectCollection.AddToWorld(pipeTopLeft);
                                break;
                            case 'Y':
                                IWorldObject pipeTopRight = new Pipe(new Vector2(position.x, 
position.y), worldObjectCollection,SpriteEngine.Instance.SpriteFactory.CreateSprite("greentopright"));
```

```
                                worldObjectCollection.AddToWorld(pipeTopRight);
                                break;

                            case 'M':
                                //These lines were removed and moved to the end so that Mario is LAST
thing that is added to the level
                                marioPosition = new Vector2(position.x, position.y);
                                break;

                            case 'K':
                                IWorldObject koopa2 = new KoopaSpawner(new Vector2(position.x,
position.y), worldObjectCollection, this);
                                worldObjectCollection.AddToWorld(koopa2);
                                break;

                            case 'E':
                                IWorldObject flagpole = new Flagpole(new Vector2(position.x,
position.y), worldObjectCollection);
                                worldObjectCollection.AddToWorld(flagpole);
                                Vector2 flagPosition = new OhioState.Graphics.Math.Vector2(position.x -
0.55f, position.y + 8.2f);
                                IWorldObject flag = new Flag(flagPosition, worldObjectCollection);
                                worldObjectCollection.AddToWorld(flag);
                                this.castleFlag = new CastleFlag(new Vector2(position.x + 8.25f,
position.y + 4), worldObjectCollection);
                                worldObjectCollection.AddToWorld((IWorldObject)castleFlag);
                                IWorldObject castle = new Castle(new Vector2(position.x + 6,
position.y), worldObjectCollection);
                                worldObjectCollection.AddToWorld(castle);
                                break;

                            case 'G':
                                IWorldObject goomba = new GoombaSpawner(new Vector2(position.x,
position.y), worldObjectCollection, this);
                                worldObjectCollection.AddToWorld(goomba);
                                break;

                            case ';':
                                position.y -= 1f;
                                position.x = 0f;
                                break;

                            case ',':
                                position.x += 1;
                                break;
                        }
                        EntityCodeFromStream = (Char)streamReader.Read();
                    }
                }
                this.mario = new Mario(this, new Vector2(marioPosition.x, marioPosition.y),
worldObjectCollection);
                worldObjectCollection.AddToWorld(mario);
                InputController = new InputControllerGameplay(this, (Mario)mario);
                playerView = new PlayerView((Mario)mario, this.coordinateFrame);
                loadingScreen = new LoadingScreen(new Vector2(0f, -.5f), worldObjectCollection);
                worldObjectCollection.AddToWorld(loadingScreen);
                headsUpDisplay = new HeadsUpDisplay(new Vector2(2f, .5f), worldObjectCollection);
                worldObjectCollection.AddToWorld(headsUpDisplay);


                //Initialize the Characters used for the timer sprites to say 400, add them in
backwards to update ones place first
                for(int iterator=2; iterator>=0; iterator--)
                {
                    AverageMario.timeSprites[iterator] = new Character(new Vector2(2f, .5f),
worldObjectCollection);
                    AverageMario.timeSprites[iterator].coordinateFrame.origin.y = 11.5f;
                    worldObjectCollection.AddToWorld(AverageMario.timeSprites[iterator]);
                }
                AverageMario.timeSprites[0].sprite =
SpriteEngine.Instance.SpriteFactory.CreateSprite("numerals2", 0, 1);
                AverageMario.timeSprites[1].sprite =
SpriteEngine.Instance.SpriteFactory.CreateSprite("numerals1", 0, 0);
                AverageMario.timeSprites[2].sprite =
SpriteEngine.Instance.SpriteFactory.CreateSprite("numerals1", 0, 0);

                //Initialize score Characters to say 000000
                for (int iterator = 0; iterator < 6; iterator++)
                {
                    AverageMario.scoreSprites[iterator] = new Character(new Vector2(2f,
.5f),worldObjectCollection);
                    AverageMario.scoreSprites[iterator].sprite =
SpriteEngine.Instance.SpriteFactory.CreateSprite("numerals1", 0, 0);
                    AverageMario.scoreSprites[iterator].coordinateFrame.origin.y = 11.5f;
                    worldObjectCollection.AddToWorld(AverageMario.scoreSprites[iterator]);
                }

                //Initialize coin counter to say 00
                AverageMario.coinTotal[0] = new Character(new Vector2(2f, .5f), worldObjectCollection);
                AverageMario.coinTotal[0].sprite =
SpriteEngine.Instance.SpriteFactory.CreateSprite("numerals1", 0, 0);
                AverageMario.coinTotal[0].coordinateFrame.origin.y = 11.5f;
                worldObjectCollection.AddToWorld(AverageMario.coinTotal[0]);
                AverageMario.coinTotal[1] = new Character(new Vector2(2f, .5f), worldObjectCollection);
                AverageMario.coinTotal[1].sprite =
SpriteEngine.Instance.SpriteFactory.CreateSprite("numerals1", 0, 0);
                AverageMario.coinTotal[1].coordinateFrame.origin.y = 11.5f;
                worldObjectCollection.AddToWorld(AverageMario.coinTotal[1]);

                //Create characters for the 1 and 1 in "world 1-1"
                worldOne = new Character(new Vector2(2f, .5f), worldObjectCollection);
                worldOne.sprite = SpriteEngine.Instance.SpriteFactory.CreateSprite("numerals1", 0, 1);
                worldOne.coordinateFrame.origin.y = 11.5f;
                worldObjectCollection.AddToWorld(worldOne);
                levelOne = new Character(new Vector2(2f, .5f), worldObjectCollection);
                levelOne.sprite = SpriteEngine.Instance.SpriteFactory.CreateSprite("numerals1", 0, 1);
                levelOne.coordinateFrame.origin.y = 11.5f;
                worldObjectCollection.AddToWorld(levelOne);
            }
            catch (Exception e)
            {
                Console.WriteLine("The file could not be read:");
                Console.WriteLine(e.Message);
            }

            /*
            this.mario = new Mario(this, new Vector2(1, 1));
            InputController = new InputControllerGameplay(this, (Mario)mario);
            playerView = new PlayerView((Mario)mario, this.coordinateFrame);
            worldObjects.Add(mario);
            */
            overallView = new OverallView(coordinateFrame);
            currentView = playerView;
        }
```

14. Give at least 3 Code Smells for the following:

```
public void Update(int multisamples = 1)
{
    //Updates the interactables, takes as many multisamples as specified.
    //If a collision occurs, calls interact, and then breaks to enable drawing.
    if (multisamples <= 0)
        return;

    float scale = 1f / multisamples;

    for (; multisamples > 0; multisamples--)
    {

        foreach (Animate x in _gameElements.OfType<Animate>())
        {
            //If x is not enabled go to next Animate
            if (!x.Enabled)
                continue;

            foreach (Interactable y in _gameElements.OfType<Interactable>())
            {

                //If y is not enabled go to next interactable
                if (!y.Enabled)
                    continue;

                //If x is the same as y, go to the next interactable

                if (x == y)
                    continue;

                //If the collision has already been checked in y's iteration, go to next
                if (CollisionAlreadyChecked(y, x))
                    continue;

                //x.Hitbox.printCoordinates();
                //y.Hitbox.printCoordinates();
                //f: 209 c: 224
                if (BoundingBox.Intersects(x.Hitbox, y.Hitbox))
                {
                    _collisions.Add(new Tuple<Interactable, Interactable>(x, y));
                    x.Interact(y);
                    y.Interact(x);
                    //this._gameForm.Invalidate();
                }
            }
        }

        _collisions.Clear();
        KeyboardState newKeyboardState = Keyboard.GetState();
        UpdateInput(newKeyboardState);
        UpdateAll(scale);
    }
}
```

15. Describe the cohesion and coupling of the following class and identify at least 4 code smells:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;


namespace JuggernautMario.Core
{
    public class BoundingBox
    {
        //A Bounding Box

        //Static methods
        public static bool Intersects(BoundingBox a, BoundingBox b)
        {
            //Check if two Bounding boxes intersect
            //return a.x <= (b.x + b.width) && (a.x + a.width) >= b.x && a.y <= (b.y + b.height) &&
(a.y + a.height) >= b.y;
            return CeilingAboveFloor(a, b) && FloorBelowCeiling(a, b) && RightGreaterThanLeft(a, b) &&
LeftLessThanRight(a, b);
        }

        public static bool FloorBelowFloor(BoundingBox a, BoundingBox b)
        {
            //check if floor of a is below floor of b
            return (a.y + e >= b.y) || (a.y - e >= b.y);
        }

        public static bool CeilingBelowCeiling(BoundingBox a, BoundingBox b)
        {
            //check if ceiling of a is below ceiling of b
            return (a.Ceiling()+ e >= b.Ceiling()) || (a.Ceiling() - e >= b.Ceiling());
        }

        public static bool FloorBelowCeiling(BoundingBox a, BoundingBox b)
        {
            //check if floor of a is below ceiling of b
            return (a.y + e >= (b.Ceiling())) || (a.y - e >= (b.Ceiling()));
        }

        public static bool CeilingBelowFloor(BoundingBox a, BoundingBox b)
        {
            //check if ceiling of a is below floor of b
            return ((a.Ceiling()) + e >= b.y) || ((a.Ceiling()) - e >= b.y);
        }

        public static bool FloorAboveFloor(BoundingBox a, BoundingBox b)
        {
            //check if floor of a is above floor of b
            return FloorBelowFloor(b, a);
        }

        public static bool CeilingAboveCeiling(BoundingBox a, BoundingBox b)
        {
            //check if ceiling of a is above ceiling of b
            return CeilingBelowCeiling(b, a);
        }

        public static bool FloorAboveCeiling(BoundingBox a, BoundingBox b)
        {
            //check if floor of a is above ceiling of b
            return CeilingBelowFloor(b, a);
        }

        public static bool CeilingAboveFloor(BoundingBox a, BoundingBox b)
        {
            //check if ceiling of a is above floor of b
            return FloorBelowCeiling(b, a);
        }

        public static bool RightLessThanRight(BoundingBox a, BoundingBox b)
        {
            //check if right edge of a is to the left of the right edge of b
            return ((a.x + a.width) + e <= (b.x + b.width)) || ((a.x + a.width) - e <= (b.x + 
b.width));
        }

        public static bool RightLessThanLeft(BoundingBox a, BoundingBox b)
        {
            //check if right edge of a is to the left of the left edge of b
            return ((a.x + a.width) + e <= b.x) || ((a.x + a.width) - e <= b.x);
        }

        public static bool LeftLessThanLeft(BoundingBox a, BoundingBox b)
        {
            //check if the left edge of a is to the left of the left edge of b
            return ((a.x + e <= b.x)) || ((a.x - e <= b.x));
        }

        public static bool LeftLessThanRight(BoundingBox a, BoundingBox b)
        {
            //check if left edge of a is to the left of right edge of b
            return a.x + e <= (b.x + b.width) || a.x - e <= (b.x + b.width);
        }

        public static bool RightGreaterThanRight(BoundingBox a, BoundingBox b)
        {
            //check if right edge of a is to the left of the right edge of b
            return RightLessThanRight(b, a);
        }

        public static bool RightGreaterThanLeft(BoundingBox a, BoundingBox b)
        {
            //check if right edge of a is to the left of the left edge of b
            return LeftLessThanRight(b, a);
        }

        public static bool LeftGreaterThanLeft(BoundingBox a, BoundingBox b)
        {
            //check if the left edge of a is to the left of the left edge of b
            return LeftLessThanLeft(b, a);
        }

        public static bool LeftGreaterThanRight(BoundingBox a, BoundingBox b)
        {
            //check if left edge of a is to the left of right edge of b
            return RightLessThanLeft(b, a);
        }

        //Instance methods
        public void Initialize(float x = 0, float y = 0, float width = 0, float height = 0)
        {
            //x and y are the coordinates for the lower left corner of the box. Width and height must 
always be positive.
            this.x = x;
            this.y = y;
            this.width = width;
            this.height = height;
        }

        public void updateOrigin(float newx, float newy)
        {
            this.x = newx;
            this.y = newy;
        }

        public void resize(float newWidth, float newHeight)
        {
            this.width = newWidth;
            this.height = newHeight;
        }

        public void printCoordinates()
        {
            System.Console.WriteLine(" F: " + this.Floor() + " C: " + this.Ceiling() + " L: " +
this.Left() + " R: " + this.Right());
        }

        public float Ceiling()
        {
            return y - height;
        }

        public float Floor()
        {
            return y;
        }

        public float Left()
        {
            return x;
        }

        public float Right()
        {
            return x + width;
        }

        public bool Intersects(BoundingBox b)
        {
            //Check if two Bounding boxes intersect
            //return a.x <= (b.x + b.width) && (a.x + a.width) >= b.x && a.y <= (b.y + b.height) &&
(a.y + a.height) >= b.y;
            return BoundingBox.Intersects(this, b);
        }

        public bool FloorBelowFloor(BoundingBox b)
        {
            //check if floor of a is below floor of b
            return BoundingBox.FloorBelowFloor(this, b);
        }

        public bool CeilingBelowCeiling(BoundingBox b)
        {
            //check if ceiling of a is below ceiling of b
            return BoundingBox.CeilingBelowCeiling(this, b);
        }

        public bool FloorBelowCeiling(BoundingBox b)
        {
```

```
        //check if floor of a is below ceiling of b
        return BoundingBox.FloorBelowCeiling(this, b);
    }

    public bool CeilingBelowFloor( BoundingBox b)
    {
        //check if ceiling of a is below floor of b
        return BoundingBox.CeilingBelowFloor(this, b);
    }

    public bool FloorAboveFloor(BoundingBox b)
    {
        //check if floor of a is above floor of b
        return BoundingBox.FloorAboveFloor(this, b);
    }

    public bool CeilingAboveCeiling(BoundingBox b)
    {
        //check if ceiling of a is above ceiling of b
        return BoundingBox.CeilingAboveCeiling(this, b);
    }

    public bool FloorAboveCeiling(BoundingBox b)
    {
        //check if floor of a is above ceiling of b
        return BoundingBox.FloorAboveCeiling(this, b);
    }

    public bool CeilingAboveFloor(BoundingBox b)
    {
        //check if ceiling of a is above floor of b
        return BoundingBox.CeilingAboveFloor(this, b);
    }

    public bool RightLessThanRight(BoundingBox b)
    {
        //check if right edge of a is to the left of the right edge of b
        return BoundingBox.RightLessThanRight(this, b);
    }

    public bool RightLessThanLeft(BoundingBox b)
    {
        //check if right edge of a is to the left of the left edge of b
        return BoundingBox.RightLessThanLeft(this, b);
    }

    public bool LeftLessThanLeft(BoundingBox b)
    {
        //check if the left edge of a is to the left of the left edge of b
        return BoundingBox.LeftLessThanLeft(this, b);
    }

    public bool LeftLessThanRight( BoundingBox b)
    {
        //check if left edge of a is to the left of right edge of b
        return BoundingBox.LeftLessThanRight(this, b);
    }

    public bool RightGreaterThanRight(BoundingBox b)
    {
        //check if right edge of a is to the left of the right edge of b
        return BoundingBox.RightGreaterThanRight(this, b);
    }
```

```
        public bool RightGreaterThanLeft(BoundingBox b)
        {
            //check if right edge of a is to the left of the left edge of b
            return BoundingBox.RightGreaterThanLeft(this, b);
        }

        public bool LeftGreaterThanLeft(BoundingBox b)
        {
            //check if the left edge of a is to the left of the left edge of b
            return BoundingBox.LeftGreaterThanLeft(this, b);
        }

        public bool LeftGreaterThanRight(BoundingBox b)
        {
            //check if left edge of a is to the left of right edge of b
            return BoundingBox.LeftGreaterThanRight(this, b);
        }

        public float x{get; set;}
        public float y{get; set;}
        public float width{get; set;}
        public float height{ get; set; }

        //error term to compensate for floating point inaccuracy
        private static float e = .0000001f;


    }
}
```

16. Define Cyclomatic complexity and give the metric for the following code:

```
public void AddToScore(int points)
{
    total += points;
}
```