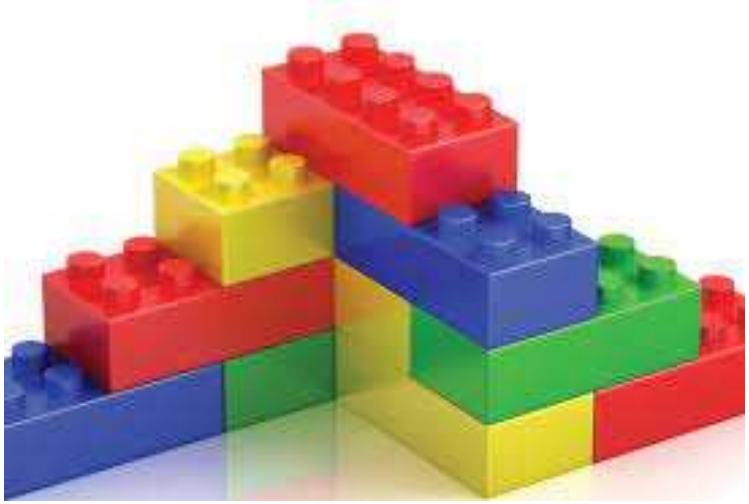
How Object Calisthenics Implement Object Oriented Principles

Sean Jennings

What are Object Calisthenics?

- A set of rules focused on maintainability, readability, testability, and comprehensibility of your code at a low level.
- The practical implementation of OO principles at a low level
- 10 rules each with a suggestion for code form



What are OO Principles?

In short

- A guide to responsibly using the pillars of OOP: encapsulation, inheritance, polymorphism and abstraction
- A description of qualities required for reusable, flexible, extensible and maintainable code at a higher level

!In short

- Encapsulate what varies
- Program to interfaces, not implementations
- Classes should be open for extension, closed for modification
- A class should only have one reason to change

Encapsulate what varies

- Segregates frequently changing code
- Allows you to extend code without affecting unchanging code

Object Calisthenics:

- 1. Only one level of indentation per method.
- 2. Don’t use the ELSE keyword.
- 4. First class collections (wrap collections in classes).
- 5. One dot per line.
- 7. Keep all entities small.
- 8. No classes with more than two instance variables.

Encapsulate what varies

```
public class DogWalker
{
    private Speed speed;

    public Speed WalkDog(Dog dog)
    {
        if (dog.Breed() == DogBreed.GermanShephard)
        {
            return Speed.Fast;
        }
        else if (dog.Breed() == DogBreed.Poodle)
        {
            return Speed.Medium;
        }
        else if (dog.Breed() == DogBreed.Pug)
        {
            return Speed.Slow;
        }
        else
        {
            return Speed.Slow;
        }
    }
}
```

```
public class DogWalker
{
    private Speed speed;
    private SpeedCalculator speedCalculator;

    public void WalkDog(Dog dog)
    {
        speed = speedCalculator.CalculateSpeed(dog);
    }
}
```

Program to interfaces, not implementations

- Allows code to change itself at runtime
- Reduces coupling and increases code flexibility

Object Calisthenics:

- 3. Wrap all primitives and strings (wrap primitive types in classes).
- 4. First class collections (wrap collections in classes).
- 5. One dot per line.
- 9. No getters/setters/properties.
- 10. All classes must have state.

Program to interfaces, not implementations

```
public class TicTacToeWithImplementation
{
    private char[,] board = new char[3, 3];

    public void Play(char player, int x, int y)
    {
        // if valid move
        board[x, y] = player;
    }
}
```

```
public class TicTacToeGameWithInterface
{
    private IBoard _board;

    public TicTacToeGameWithInterface()
    {
        _board = new Board();
    }

    public void Play(Player player, Position position)
    {
        _board.Play(player, position);
    }
}
```

Classes should be open for extension, closed for modification

- We allow extension of methods to add new behaviour
- We don’t allow modification of existing code which may introduce bugs

Object Calisthenics:

- 3. Wrap all primitives and strings (wrap primitive types in classes).
- 4. First class collections (wrap collections in classes).
- 9. No getters/setters/properties.
- 10. All classes must have state.

Classes should be open for extension, closed for modification

```
public class NuclearReactor
{
    public int Temperature { get; set; } = 1100;

    public void CalculateTemperature()
    {
        // logic to calculate temperature
        Shutdown();
    }

    private void Shutdown()
    {
        if (Temperature > 1000)
        {
            throw new Exception("Reactor failure");
        }
    }
}
```

```
public class NuclearReactor
{
    private readonly int _temperature;

    public void CalculateTemperature()
    {
        // logic to calculate temperature
        Shutdown();
    }

    private void Shutdown()
    {
        if (_temperature > 1000)
        {
            throw new Exception("Reactor failure");
        }
        AdditionalShutdownLogic();

    }

    protected virtual void AdditionalShutdownLogic() {}
}
```

A class should only have one reason to change

- Change increases maintenance and development cost
- Separating responsibility removes reasons for changing

Object Calisthenics:

- All of them!

A class should only have one reason to change

```
public class Engine
{
    private int FuelLevel { get; set; }
    private int EngineSpeed { get; set; }
    private int magicalNumber = 2;
    private int temp { get; set; }

    public void Start()
    {
        if (FuelLevel < 10)
        {
            // process low fuel injection
        }
        else
        {
            // process high fuel injection
        }

        if (temp > 100)
        {
            EngineSpeed = FuelLevel * 10 / magicalNumber;
            if (FuelLevel < 10)
            {
                // process high temp low fuel injection
            }
        }
    }
}
```

```
public class Engine
{
    private EngineProcessor _engineProcessor;
    public void Start()
    {
        _engineProcessor.Start();
        // other related logic
    }
}
```

Where next?

- Object calisthenics provide a guide to clean code at low level
- Object calisthenics don't provide guide to code at high level
- Design patterns are high-level concepts that apply OO principles

QUESTIONS?

THANK YOU

Sean Jennings

sean.jennings@fdbhealth.com