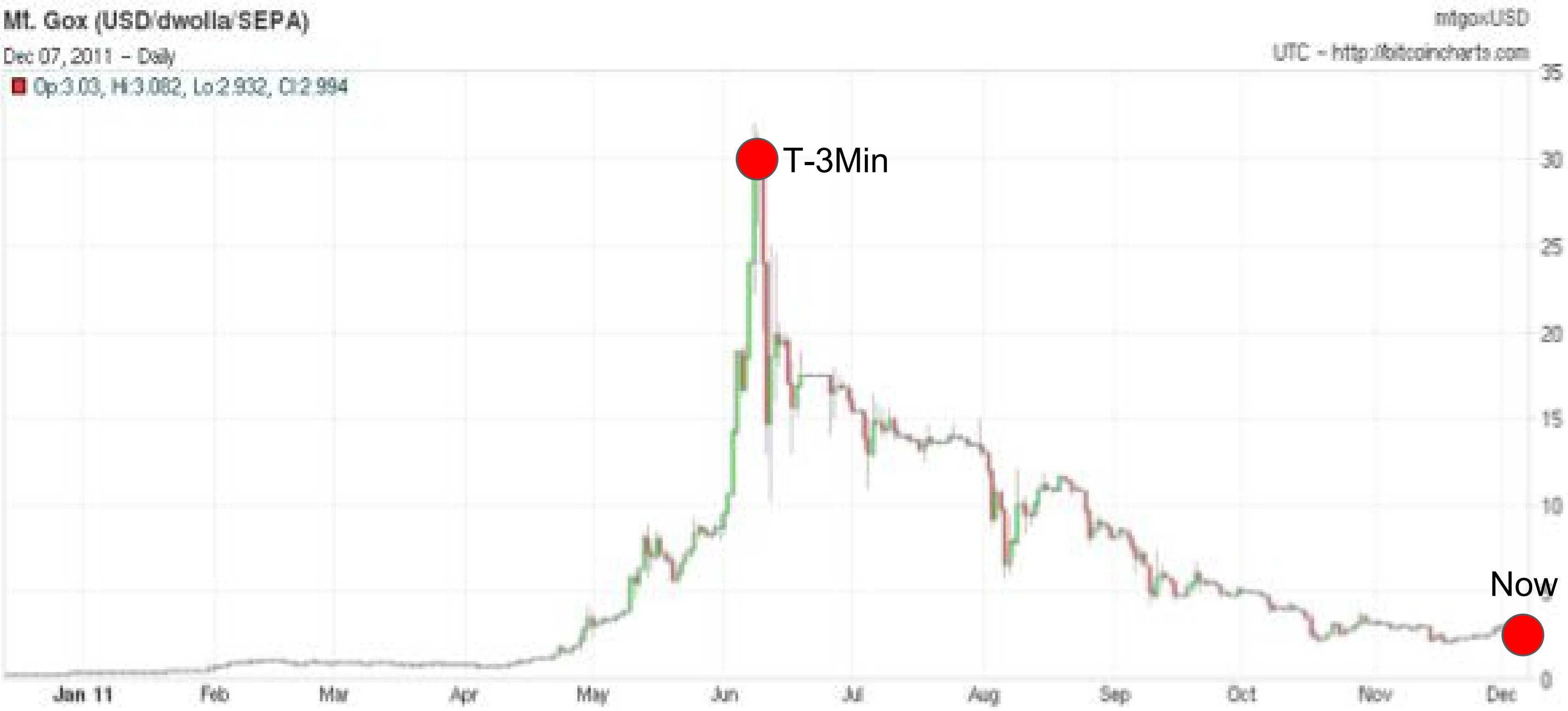
An introduction to the magical land of SOLID and Dependency Injection and IoC and Unit Testing and other really cool stuff

Some Context (about me) ‘cause I’m self-important

- Coding for >10 years
  - I’ve coded a lot!
- 20 Years Old
  - Only a bit of industry experience!
- Have only been doing DI/IOC/Testing for ~2 years.
  - By no means an expert!

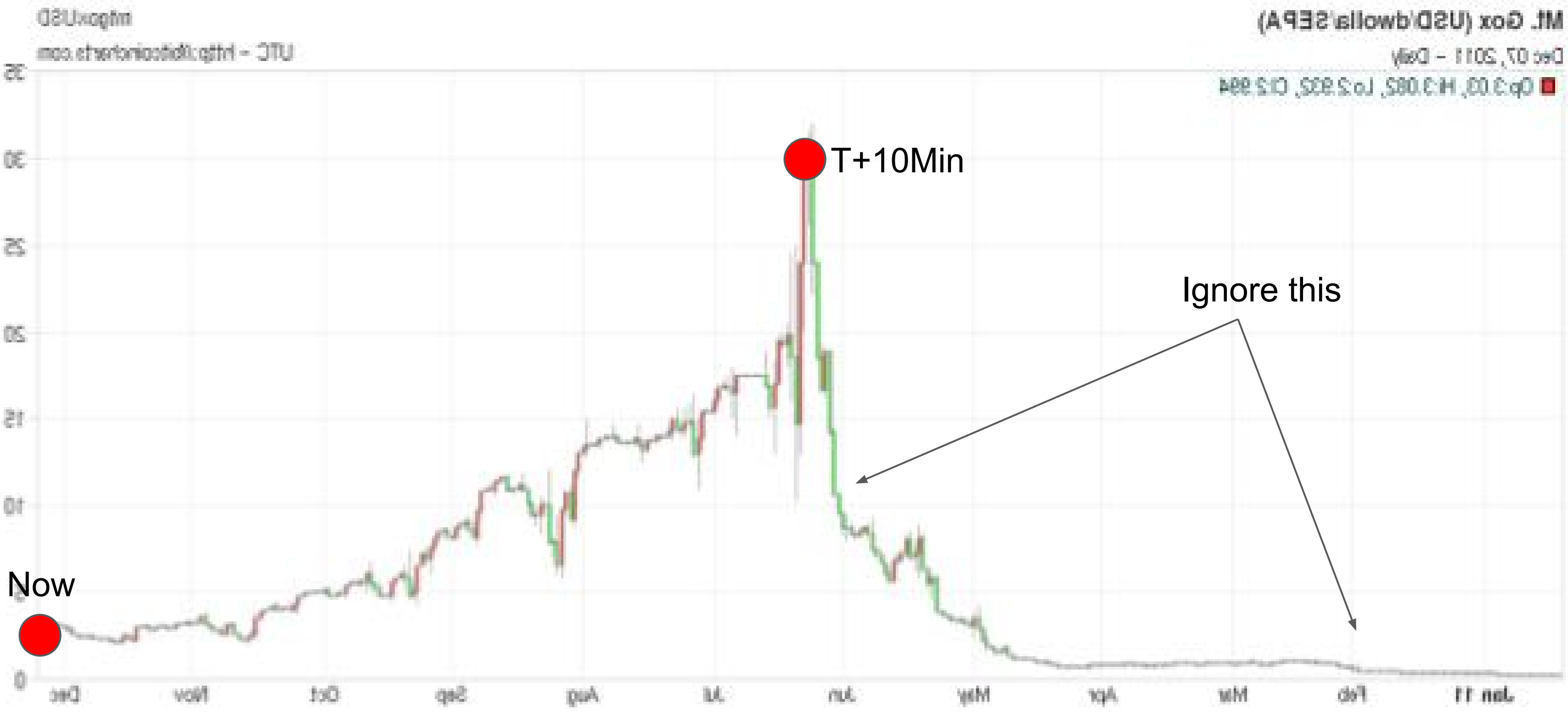
Your Expectations



T-3Min

Now

Where I hope we go from here



T+10Min

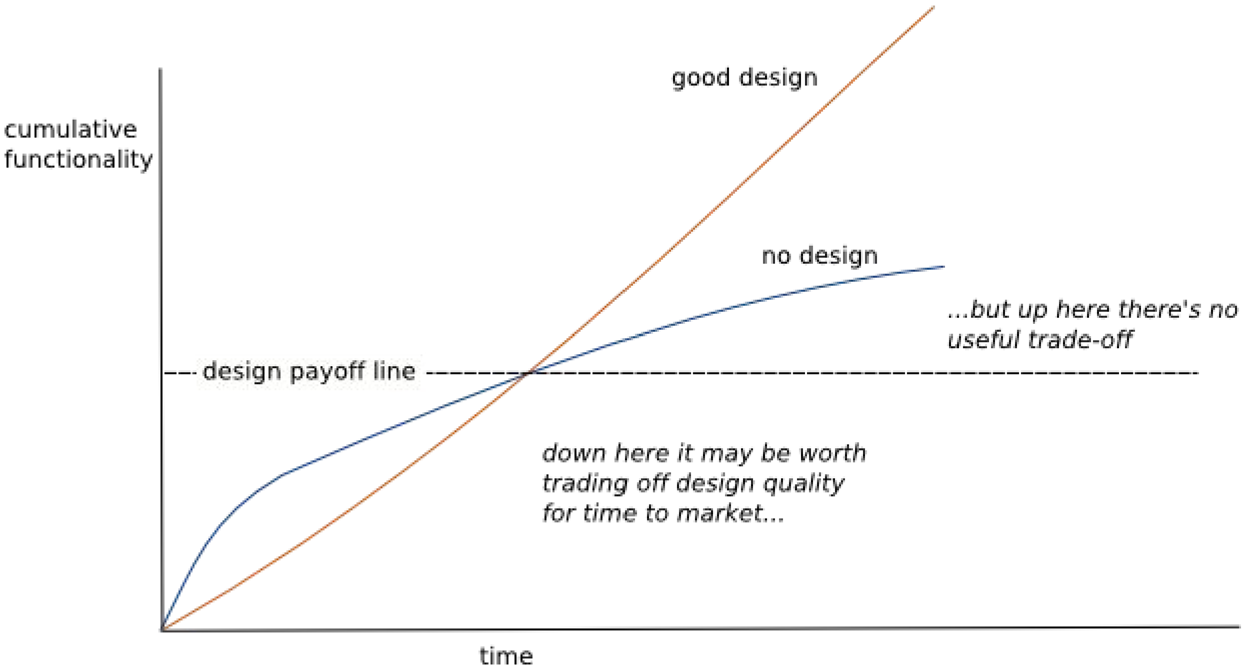
Ignore this

Now

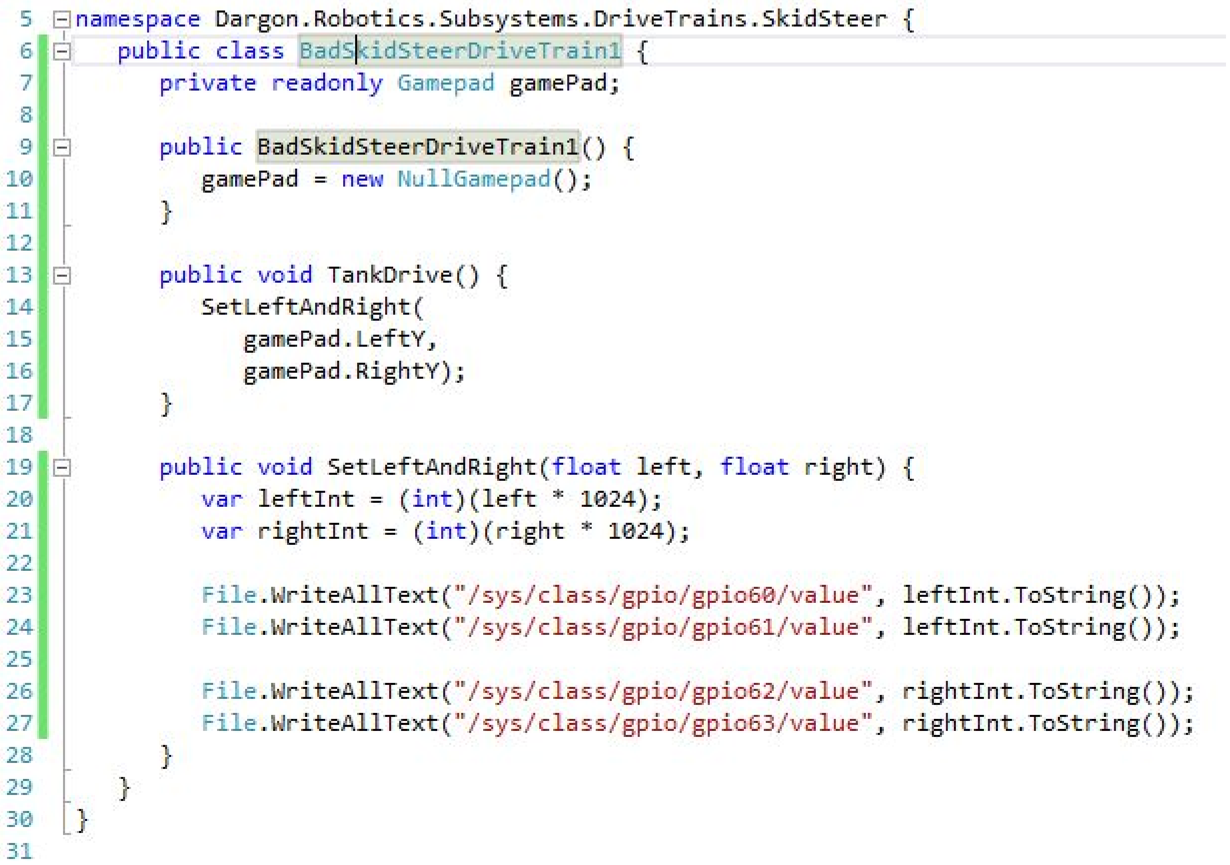
Goals

- Develop informed voices that can counter mine.
- Grow teammates through shared knowledge.
- Understand no code is “good”
    - Tradeoffs between “beautiful” code and getting things done.

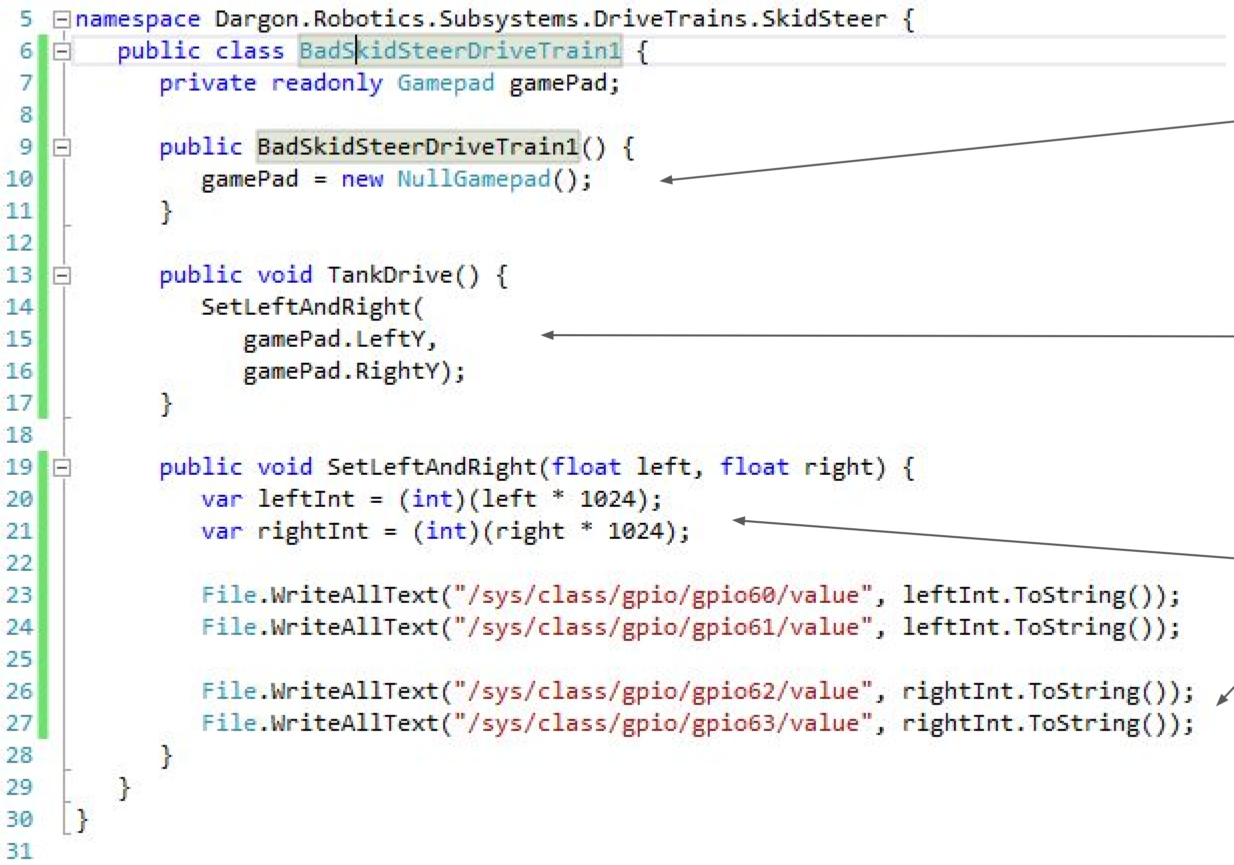
Hacking together code is sometimes OK.



SOLID Principles



```
namespace Dargon.Robotics.Subsystems.DriveTrains.SkidSteer {
   public class BadSkidSteerDriveTrain1 {
      private readonly Gamepad gamePad;

      public BadSkidSteerDriveTrain1() {
         gamePad = new NullGamepad();
      }

      public void TankDrive() {
         SetLeftAndRight(
            gamePad.LeftY,
            gamePad.RightY);
      }

      public void SetLeftAndRight(float left, float right) {
         var leftInt = (int)(left * 1024);
         var rightInt = (int)(right * 1024);

         File.WriteAllText("/sys/class/gpio/gpio60/value", leftInt.ToString());
         File.WriteAllText("/sys/class/gpio/gpio61/value", leftInt.ToString());

         File.WriteAllText("/sys/class/gpio/gpio62/value", rightInt.ToString());
         File.WriteAllText("/sys/class/gpio/gpio63/value", rightInt.ToString());
      }
   }
}
```



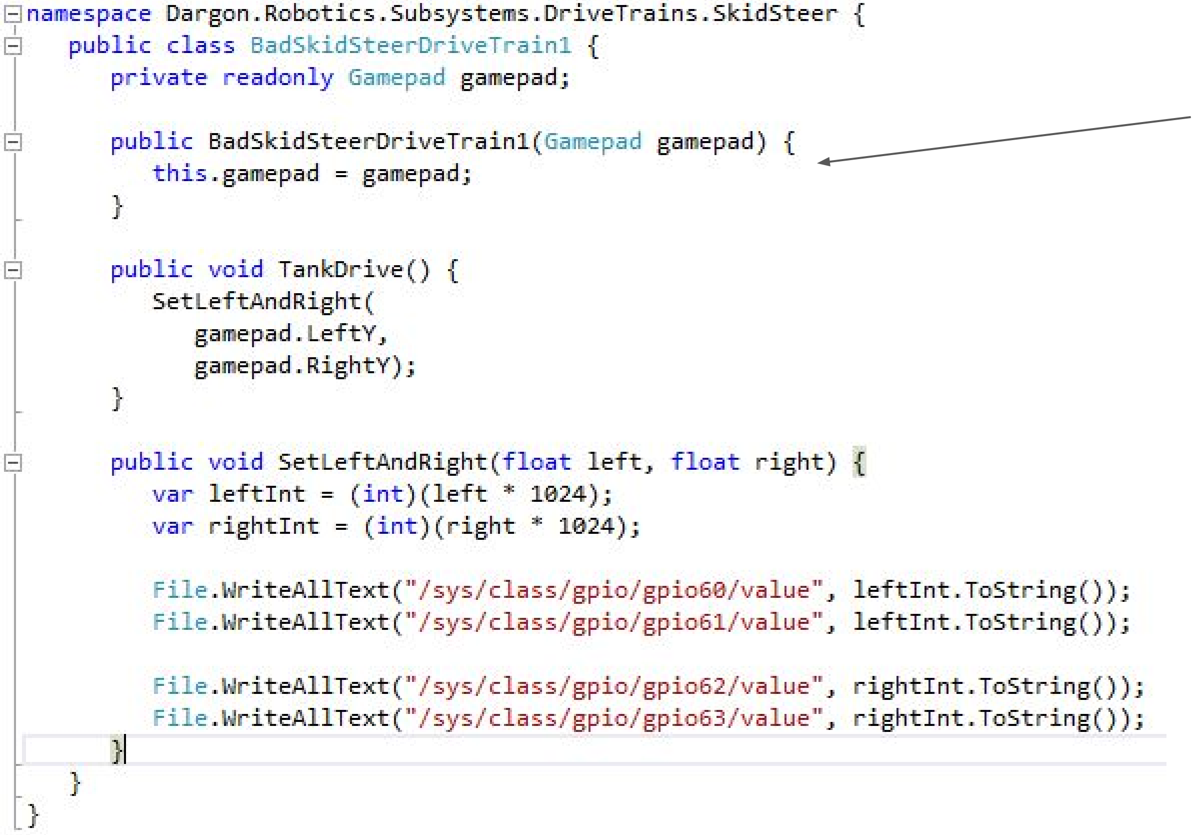
```
namespace Dargon.Robotics.Subsystems.DriveTrains.SkidSteer {
   public class BadSkidSteerDriveTrain1 {
      private readonly Gamepad gamePad;

      public BadSkidSteerDriveTrain1() {
         gamePad = new NullGamepad();
      }

      public void TankDrive() {
         SetLeftAndRight(
            gamePad.LeftY,
            gamePad.RightY);
      }

      public void SetLeftAndRight(float left, float right) {
         var leftInt = (int)(left * 1024);
         var rightInt = (int)(right * 1024);

         File.WriteAllText("/sys/class/gpio/gpio60/value", leftInt.ToString());
         File.WriteAllText("/sys/class/gpio/gpio61/value", leftInt.ToString());

         File.WriteAllText("/sys/class/gpio/gpio62/value", rightInt.ToString());
         File.WriteAllText("/sys/class/gpio/gpio63/value", rightInt.ToString());
      }
   }
}
```

Type must know implementations of its dependencies!

Testing TankDrive involves testing SetLeftAndRight!

BeagleBone specific!

SOLID - Dependency Injection (DI)

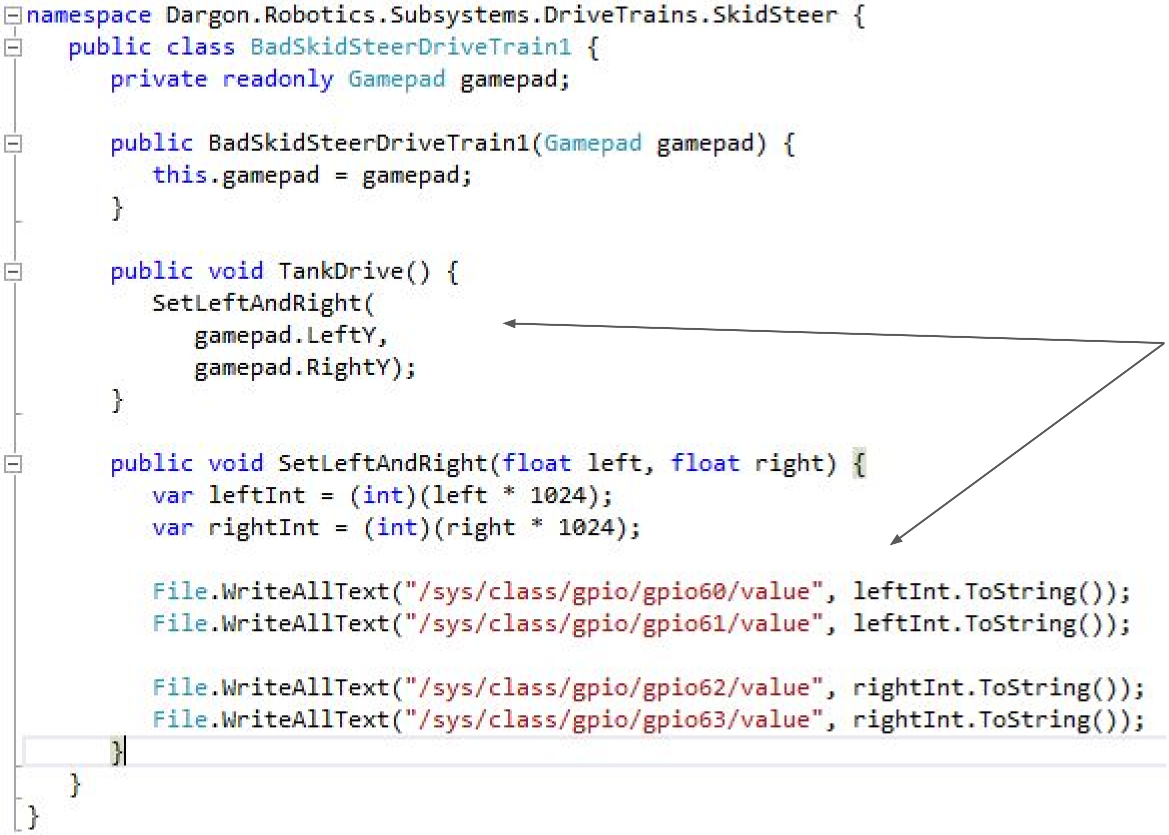


```
namespace Dargon.Robotics.Subsystems.DriveTrains.SkidSteer {
   public class BadSkidSteerDriveTrain1 {
      private readonly Gamepad gamepad;

      public BadSkidSteerDriveTrain1(Gamepad gamepad) {
         this.gamepad = gamepad;
      }

      public void TankDrive() {
         SetLeftAndRight(
            gamepad.LeftY,
            gamepad.RightY);
      }

      public void SetLeftAndRight(float left, float right) {
         var leftInt = (int)(left * 1024);
         var rightInt = (int)(right * 1024);

         File.WriteAllText("/sys/class/gpio/gpio60/value", leftInt.ToString());
         File.WriteAllText("/sys/class/gpio/gpio61/value", leftInt.ToString());

         File.WriteAllText("/sys/class/gpio/gpio62/value", rightInt.ToString());
         File.WriteAllText("/sys/class/gpio/gpio63/value", rightInt.ToString());
      }
   }
}
```

Inject dependencies at ctor!

- Reusable component
- Configured by “higher being”
- Allows our object to be “dumber” and think more about interfaces.

SOLID - Single Responsibility Principle (SRP)



```
namespace Dargon.Robotics.Subsystems.DriveTrains.SkidSteer {
   public class BadSkidSteerDriveTrain1 {
      private readonly Gamepad gamepad;

      public BadSkidSteerDriveTrain1(Gamepad gamepad) {
         this.gamepad = gamepad;
      }

      public void TankDrive() {
         SetLeftAndRight(
            gamepad.LeftY,
            gamepad.RightY);
      }

      public void SetLeftAndRight(float left, float right) {
         var leftInt = (int)(left * 1024);
         var rightInt = (int)(right * 1024);

         File.WriteAllText("/sys/class/gpio/gpio60/value", leftInt.ToString());
         File.WriteAllText("/sys/class/gpio/gpio61/value", leftInt.ToString());

         File.WriteAllText("/sys/class/gpio/gpio62/value", rightInt.ToString());
         File.WriteAllText("/sys/class/gpio/gpio63/value", rightInt.ToString());
      }
   }
}
```

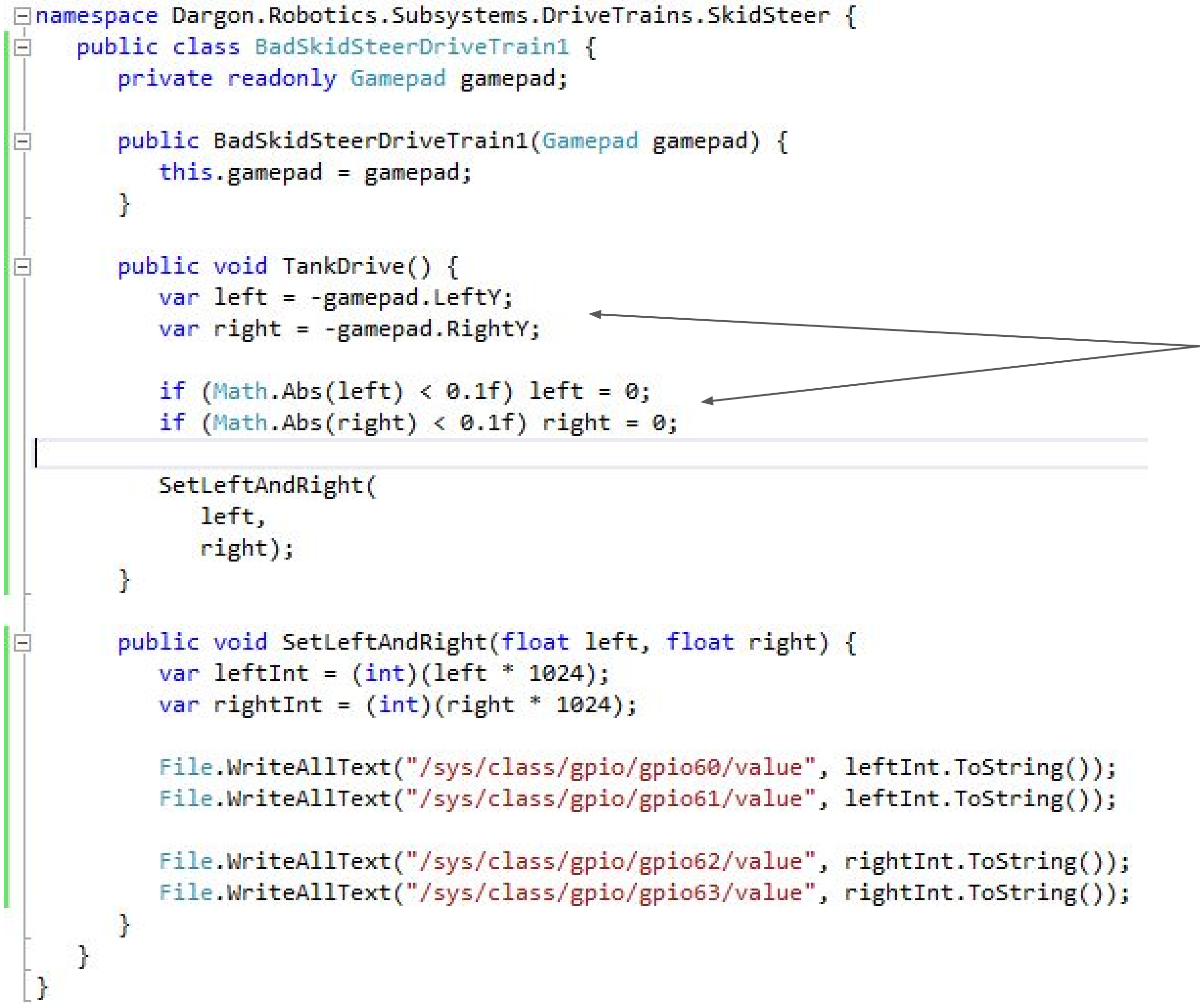
Class does too much work!

- Multiple Responsibilities - can make difficult to reason about.
- Not Modular!

Responsibilities:

- Converting joy state to motion commands.
- Writing motion commands to actual system.

Tight Coupling



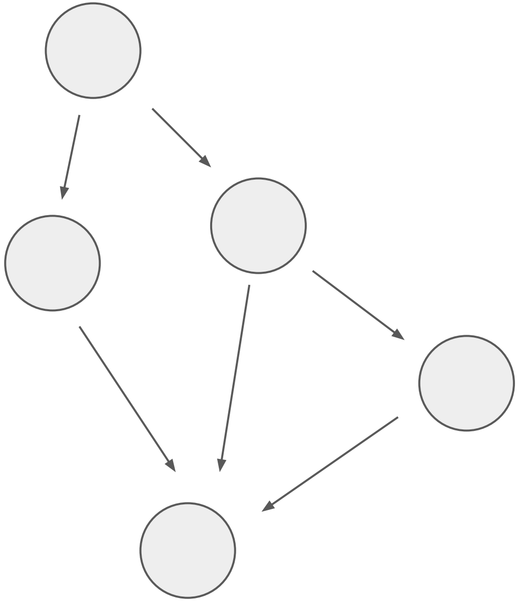
```
namespace Dargon.Robotics.Subsystems.DriveTrains.SkidSteer {
   public class BadSkidSteerDriveTrain1 {
      private readonly Gamepad gamepad;

      public BadSkidSteerDriveTrain1(Gamepad gamepad) {
         this.gamepad = gamepad;
      }

      public void TankDrive() {
         var left = -gamepad.LeftY;
         var right = -gamepad.RightY;

         if (Math.Abs(left) < 0.1f) left = 0;
         if (Math.Abs(right) < 0.1f) right = 0;

         SetLeftAndRight(
            left,
            right);
      }

      public void SetLeftAndRight(float left, float right) {
         var leftInt = (int)(left * 1024);
         var rightInt = (int)(right * 1024);

         File.WriteAllText("/sys/class/gpio/gpio60/value", leftInt.ToString());
         File.WriteAllText("/sys/class/gpio/gpio61/value", leftInt.ToString());

         File.WriteAllText("/sys/class/gpio/gpio62/value", rightInt.ToString());
         File.WriteAllText("/sys/class/gpio/gpio63/value", rightInt.ToString());
      }
   }
}
```

Minor Example:

- DriveTrain working on too many concepts!
  - Flipped Axis
  - Deadzone
- DT working around intricacies of GamePad itself!

Sort of a bad example, but, time constraints, you know.

SOLID - Open/Closed Principle (OCP)

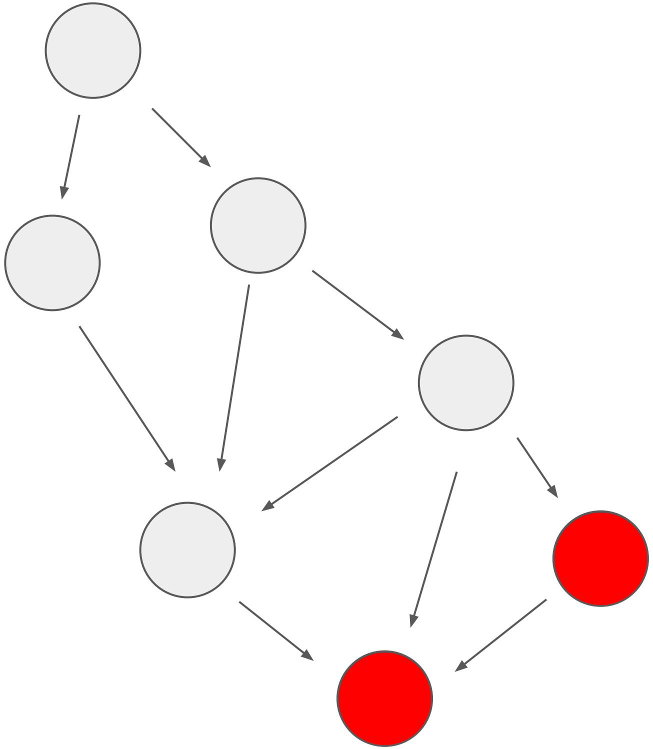


“software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification”

- Has a billion different conflicting interpretations.

Interpretation: “Holy Grail” of OOP.

S**O**LID - Open/Closed Principle (OCP)

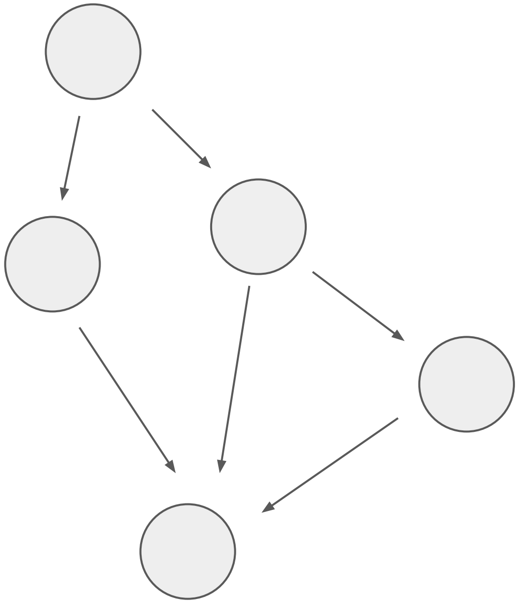


“software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification”

- Has a billion different conflicting interpretations.

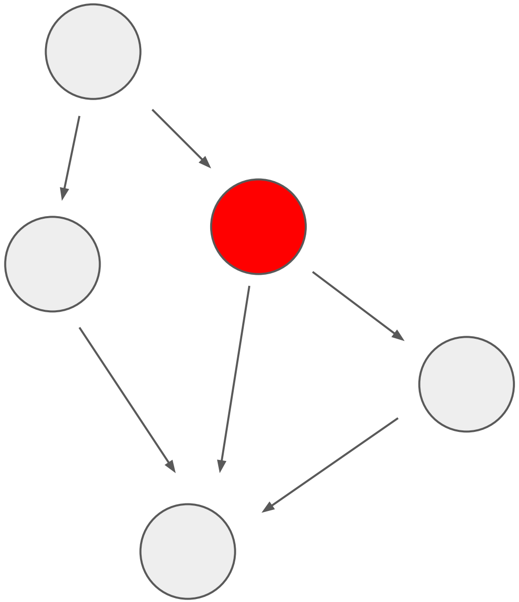
Interpretation: “Holy Grail” of OOP.

SOLID - Open/Closed Principle (OCP)



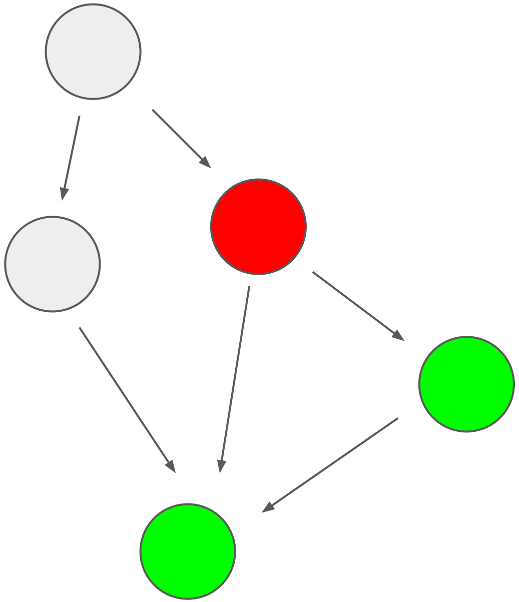
Rationale: Modifying upward dependencies can cascade downward to consumers.

SOLID - Open/Closed Principle (OCP)



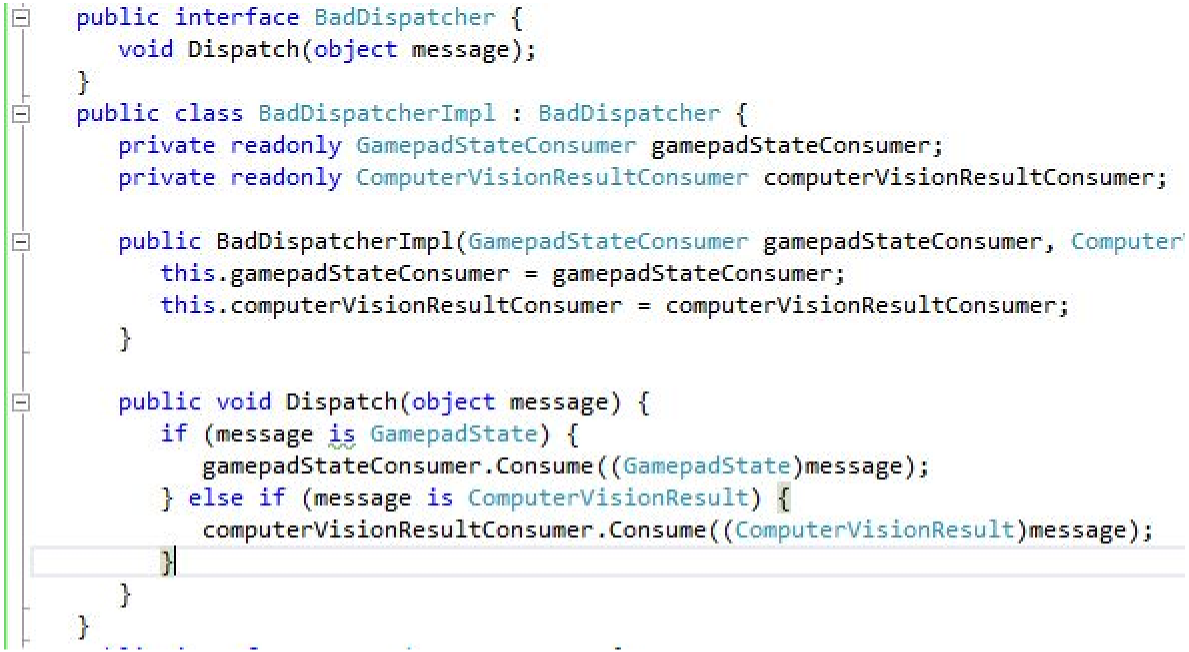
Rationale: Modifying upward dependencies can cascade downward to consumers.

SOLID - Open/Closed Principle (OCP)



Rationale: Modifying upward dependencies can cascade downward to consumers.

SOLID - Open/Closed Principle (OCP)



```
public interface BadDispatcher {
   void Dispatch(object message);
}
public class BadDispatcherImpl : BadDispatcher {
   private readonly GamepadStateConsumer gamepadStateConsumer;
   private readonly ComputerVisionResultConsumer computerVisionResultConsumer;

   public BadDispatcherImpl(GamepadStateConsumer gamepadStateConsumer, Computer
      this.gamepadStateConsumer = gamepadStateConsumer;
      this.computerVisionResultConsumer = computerVisionResultConsumer;
   }

   public void Dispatch(object message) {
      if (message is GamepadState) {
         gamepadStateConsumer.Consume((GamepadState)message);
      } else if (message is ComputerVisionResult) {
         computerVisionResultConsumer.Consume((ComputerVisionResult)message);
      }
   }
}
```

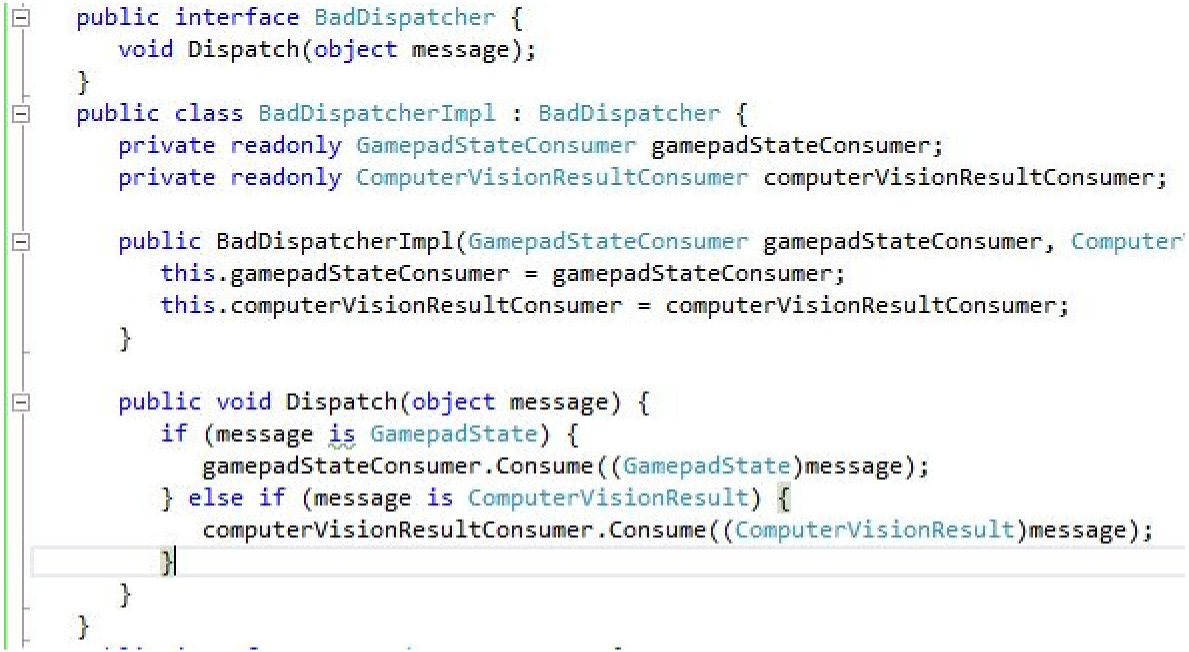
Trivial Case - How do you add new cases?

- Add new `if` case!

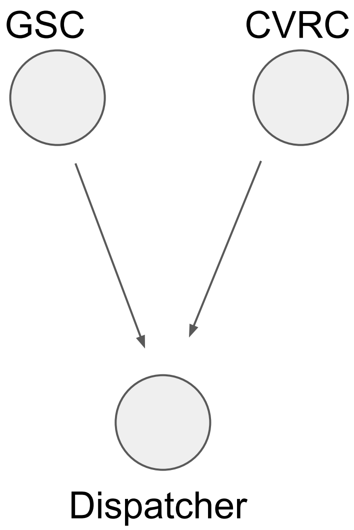
Problem:

- What if dispatcher code in another library?
  - Update other library, recompile!

SOLID - Open/Closed Principle (OCP)



```
public interface BadDispatcher {
   void Dispatch(object message);
}
public class BadDispatcherImpl : BadDispatcher {
   private readonly GamepadStateConsumer gamepadStateConsumer;
   private readonly ComputerVisionResultConsumer computerVisionResultConsumer;

   public BadDispatcherImpl(GamepadStateConsumer gamepadStateConsumer, Computer
      this.gamepadStateConsumer = gamepadStateConsumer;
      this.computerVisionResultConsumer = computerVisionResultConsumer;
   }

   public void Dispatch(object message) {
      if (message is GamepadState) {
         gamepadStateConsumer.Consume((GamepadState)message);
      } else if (message is ComputerVisionResult) {
         computerVisionResultConsumer.Consume((ComputerVisionResult)message);
      }
   }
}
```

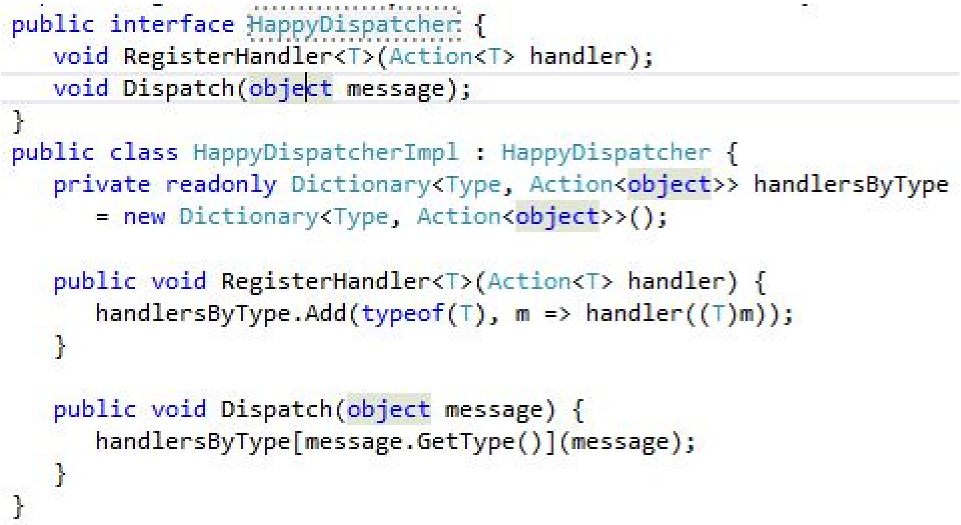


GSC

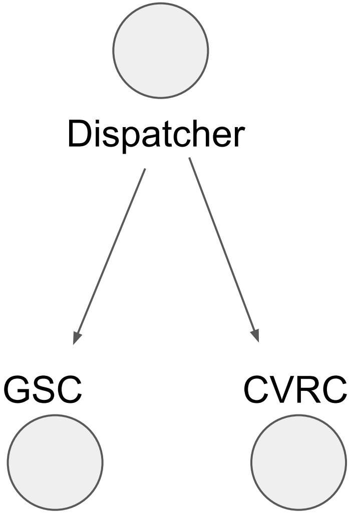
CVRC

Dispatcher

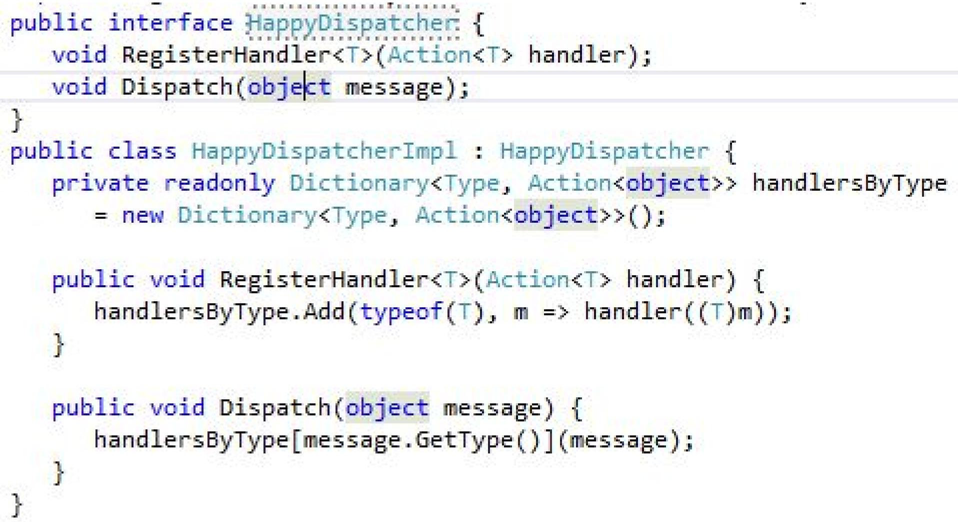
SOLID - Open/Closed Principle (OCP)



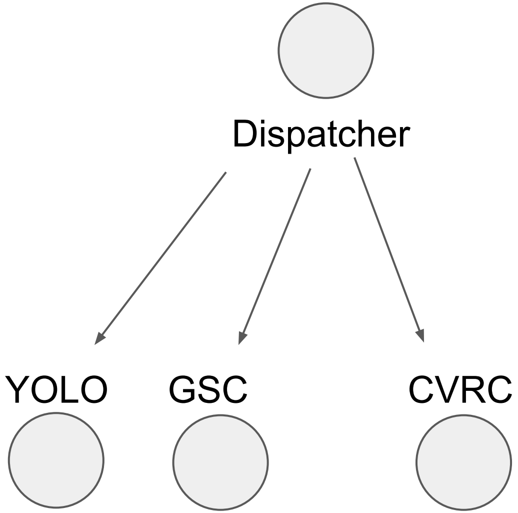
```
public interface HappyDispatcher {
   void RegisterHandler<T>(Action<T> handler);
   void Dispatch(object message);
}
public class HappyDispatcherImpl : HappyDispatcher {
   private readonly Dictionary<Type, Action<object>> handlersByType
      = new Dictionary<Type, Action<object>>();

   public void RegisterHandler<T>(Action<T> handler) {
      handlersByType.Add(typeof(T), m => handler((T)m));
   }

   public void Dispatch(object message) {
      handlersByType[message.GetType()](message);
   }
}
```



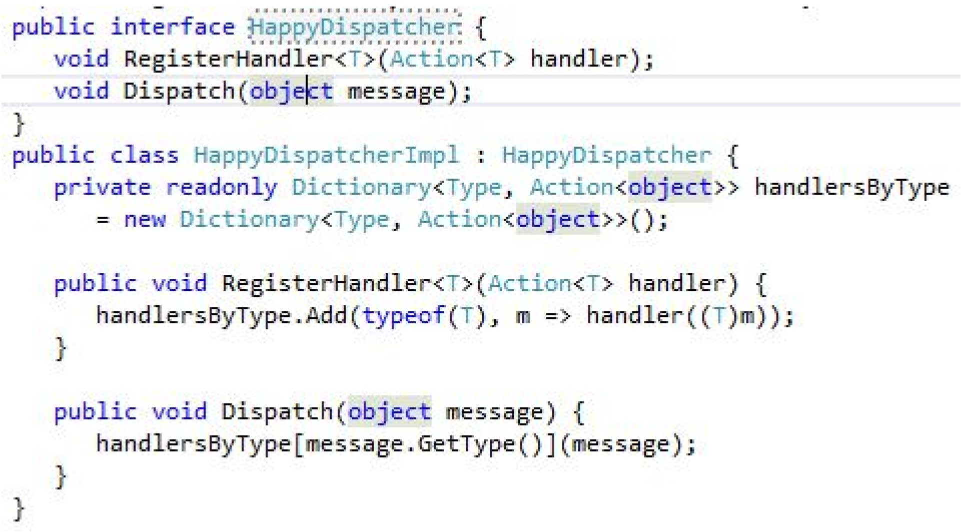
SOLID - Open/Closed Principle (OCP)



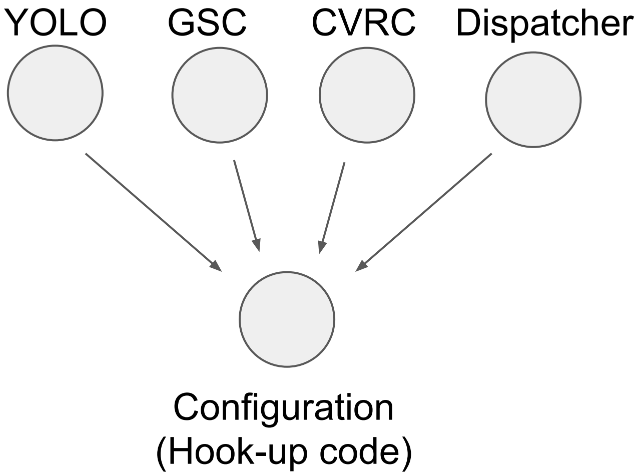
```
public interface HappyDispatcher {
   void RegisterHandler<T>(Action<T> handler);
   void Dispatch(object message);
}
public class HappyDispatcherImpl : HappyDispatcher {
   private readonly Dictionary<Type, Action<object>> handlersByType
      = new Dictionary<Type, Action<object>>();

   public void RegisterHandler<T>(Action<T> handler) {
      handlersByType.Add(typeof(T), m => handler((T)m));
   }

   public void Dispatch(object message) {
      handlersByType[message.GetType()](message);
   }
}
```



SOLID - Open/Closed Principle (OCP)



```
public interface HappyDispatcher {
   void RegisterHandler<T>(Action<T> handler);
   void Dispatch(object message);
}
public class HappyDispatcherImpl : HappyDispatcher {
   private readonly Dictionary<Type, Action<object>> handlersByType
      = new Dictionary<Type, Action<object>>();

   public void RegisterHandler<T>(Action<T> handler) {
      handlersByType.Add(typeof(T), m => handler((T)m));
   }

   public void Dispatch(object message) {
      handlersByType[message.GetType()](message);
   }
}
```



YOLO

GSC

CVRC

Dispatcher

Configuration
(Hook-up code)

SOLID - Liskov Substitution Principle (LSP)

- Program to abstractions, not concretions.
  - Classical OOP: If A:B, A is a B.
  - LSP's interpretation: If A:B, A is a substitute for B
- Classic Example

  -
    ```
    class Rectangle {
       public virtual int Width { get; set; }
       public virtual int Height { get; set; }
    }
    ```

SO**L**ID - Liskov Substitution Principle (LSP)

- Program to abstractions, not concretions.
  - Classical OOP: If A:B, A is a B.
  - LSP’s interpretation: If A:B, A is a substitute for B
- Classic Example
  -
    ```
    class Rectangle {
       public virtual int Width { get; set; }
       public virtual int Height { get; set; }
    }
    ```

- Don’t do this.
  -
    ```
    class Square : Rectangle {
       private int width, height;
       public override int Width {
          get { return width; }
          set { width = value;
                height = value; }
       }
       public override int Height {
          get { return height; }
          set { width = value;
                height = value; }
       }
    ```

SO**L**ID - Liskov Substitution Principle (LSP)

- Program to abstractions, not concretions.
  - Classical OOP: If A:B, A is a B.
  - LSP's interpretation: If A:B, A is a substitute for B
- Classic Example

  -
    ```
    class Rectangle {
       public virtual int Width { get; set; }
       public virtual int Height { get; set; }
    }
    ```
- Because this happens:

  -
    ```
    void Scale(this Rectangle r, float scale) {
       r.Width *= scale; r.Height *= scale;
    }
    ```
  - `new Rectangle { W = 10, H = 20 }.Scale(2); // {20, 40}`
  - `new Square { W = 10, H = 10 }.Scale(2); // {40, 40}`

- Don't do this.

  -
    ```
    class Square : Rectangle {
       private int width, height;
       public override int Width {
          get { return width; }
          set { width = value;
                height = value; }
       }
       public override int Height {
          get { return height; }
          set { width = value;
                height = value; }
       }
    ```

SOLID - Liskov Substitution Principle (LSP)

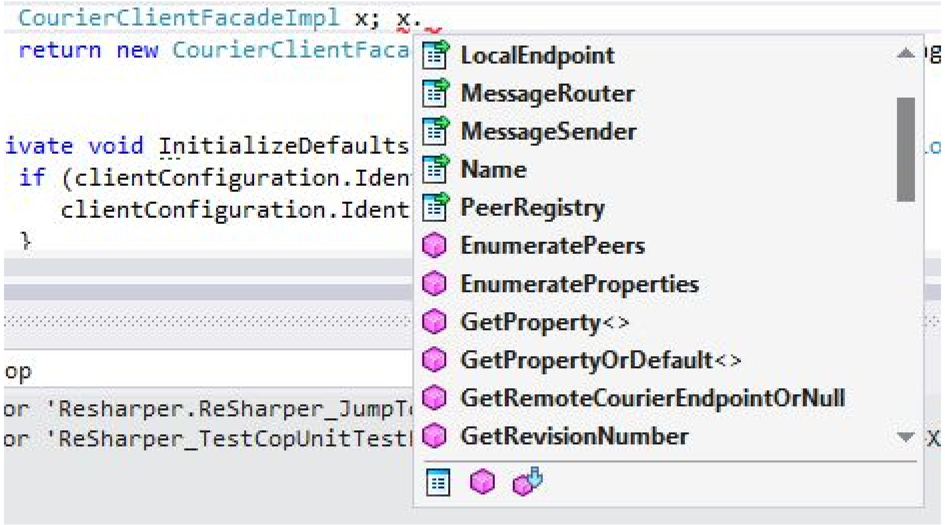
- If A:B, A is a substitute for B.
  - You can reason about “A” as if it were a “B”.
    - All guarantees about “B” apply to “A”.
    - (Of course, not guarantees about “A” apply to B)
  - Ties very well into Interface Segregation Principle

SOLID - Interface Segregation Principle (ISP)

- “No client should be forced to depend on methods it does not use.”
- Solution: Control the scope of your interfaces, have multiple interfaces.
  - Think of as “Privilege levels” in apps!
- Why it matters:
  - Dependency control.
  - Ability to reason about code.

SOLID - Interface Segregation Principle (ISP)

- “No client should be forced to depend on methods it does not use.”

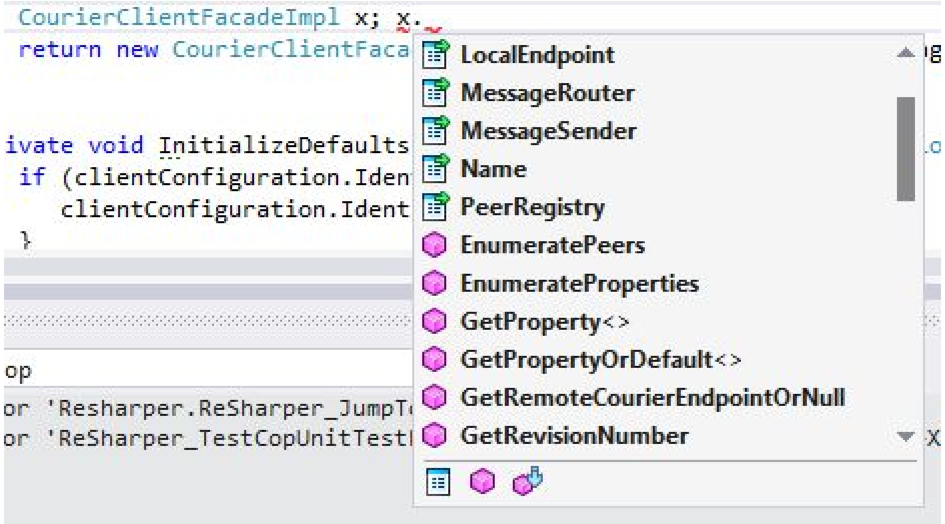


```
 CourierClientFacadeImpl x; x.
 return new CourierClientFaca

ivate void InitializeDefaults
 if (clientConfiguration.Iden
    clientConfiguration.Ident
 }
```

```
op
or 'Resharper.ReSharper_JumpT
or 'ReSharper_TestCopUnitTest
```

SOLID - Interface Segregation Principle (ISP)

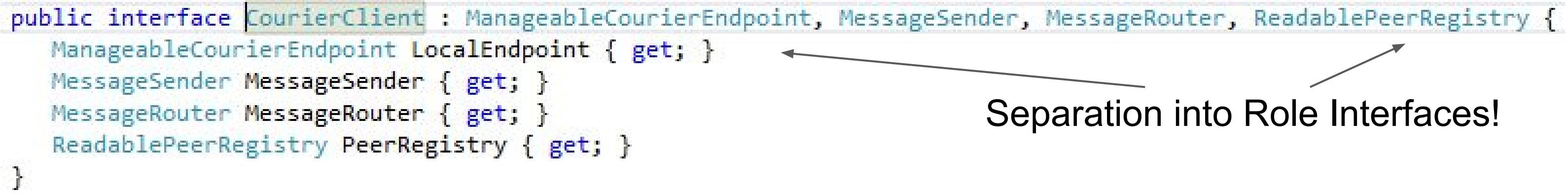
- “No client should be forced to depend on methods it does not use.”



```
CourierClientFacadeImpl x; x.
return new CourierClientFaca                          g

ivate void InitializeDefaults                         o
 if (clientConfiguration.Iden
    clientConfiguration.Ident
 }

op
or 'Resharper.ReSharper_JumpT
or 'ReSharper_TestCopUnitTest                         X
```

Interface Broken Up!

- Node Identity
- Message Sender
- Message Router
- Peer Discovery/Lookup



```
public interface CourierClient : ManageableCourierEndpoint, MessageSender, MessageRouter, ReadablePeerRegistry {
   ManageableCourierEndpoint LocalEndpoint { get; }
   MessageSender MessageSender { get; }
   MessageRouter MessageRouter { get; }
   ReadablePeerRegistry PeerRegistry { get; }
}
```

Separation into Role Interfaces!

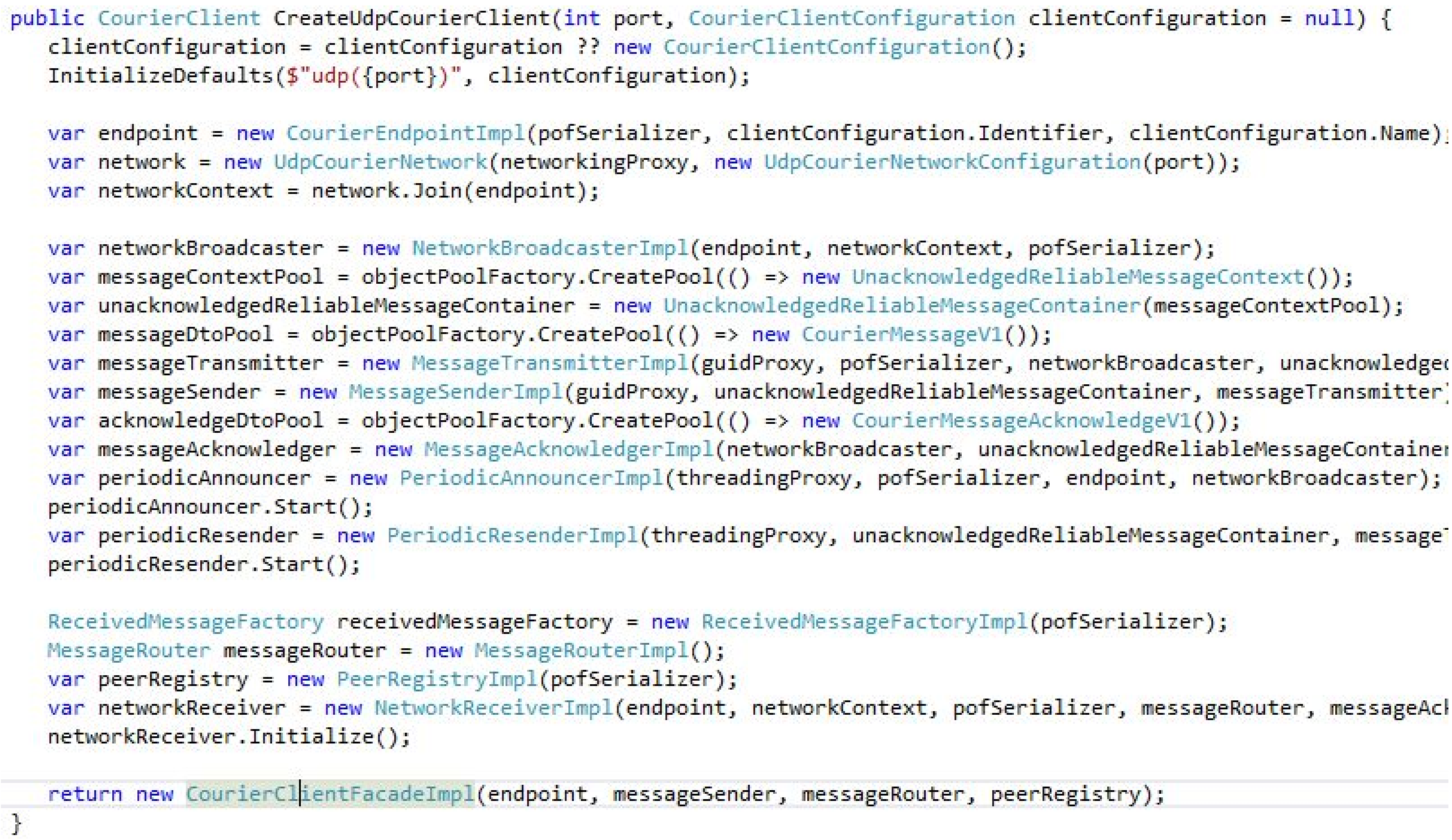
SOLID Review!

- **S**ingle Responsibility Principle
- **O**pen/Closed Principle
- **L**iskov Substitution Principle
- **I**nterface Segregation Principle
- **D**ependency Injection

Problem: Object Creation is a Responsibility

- As we’ll learn later: new T() difficult to unit test
- Solution - Factories
  - Factories: Objects that are responsible for creating other objects

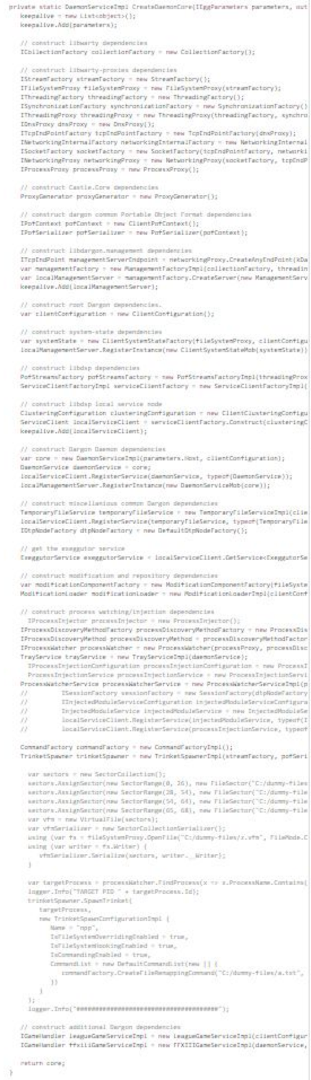
  -
    ```
    class XFactory {
      private DepA depA; private DepB depB;
      .ctor()
      public X CreateX() {
        var depC = new DepC(depA);
        return new X(depB, depC);
      }
    }
    ```



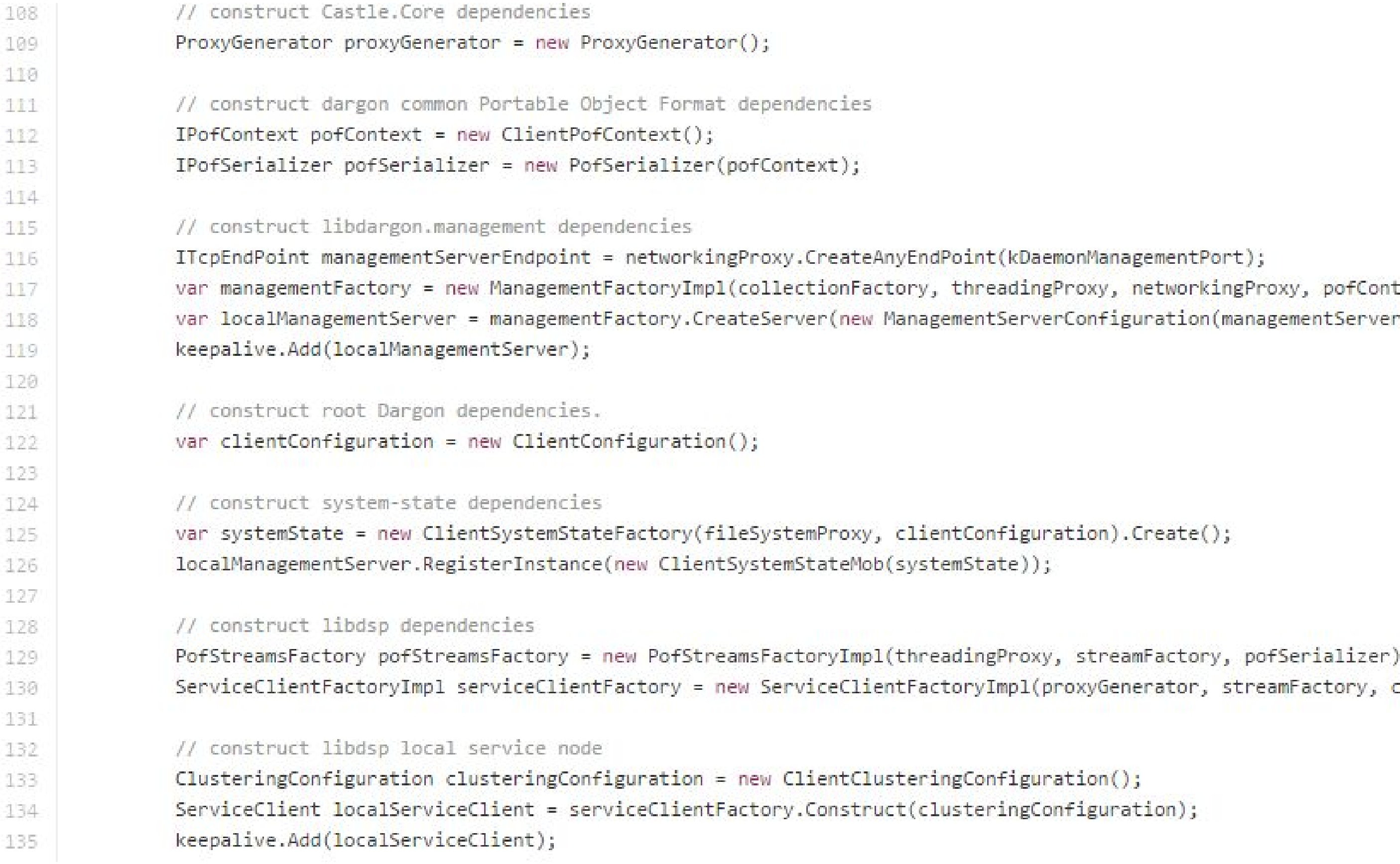
```
public CourierClient CreateUdpCourierClient(int port, CourierClientConfiguration clientConfiguration = null) {
   clientConfiguration = clientConfiguration ?? new CourierClientConfiguration();
   InitializeDefaults($"udp({port})", clientConfiguration);

   var endpoint = new CourierEndpointImpl(pofSerializer, clientConfiguration.Identifier, clientConfiguration.Name);
   var network = new UdpCourierNetwork(networkingProxy, new UdpCourierNetworkConfiguration(port));
   var networkContext = network.Join(endpoint);

   var networkBroadcaster = new NetworkBroadcasterImpl(endpoint, networkContext, pofSerializer);
   var messageContextPool = objectPoolFactory.CreatePool(() => new UnacknowledgedReliableMessageContext());
   var unacknowledgedReliableMessageContainer = new UnacknowledgedReliableMessageContainer(messageContextPool);
   var messageDtoPool = objectPoolFactory.CreatePool(() => new CourierMessageV1());
   var messageTransmitter = new MessageTransmitterImpl(guidProxy, pofSerializer, networkBroadcaster, unacknowledged
   var messageSender = new MessageSenderImpl(guidProxy, unacknowledgedReliableMessageContainer, messageTransmitter)
   var acknowledgeDtoPool = objectPoolFactory.CreatePool(() => new CourierMessageAcknowledgeV1());
   var messageAcknowledger = new MessageAcknowledgerImpl(networkBroadcaster, unacknowledgedReliableMessageContainer
   var periodicAnnouncer = new PeriodicAnnouncerImpl(threadingProxy, pofSerializer, endpoint, networkBroadcaster);
   periodicAnnouncer.Start();
   var periodicResender = new PeriodicResenderImpl(threadingProxy, unacknowledgedReliableMessageContainer, messageT
   periodicResender.Start();

   ReceivedMessageFactory receivedMessageFactory = new ReceivedMessageFactoryImpl(pofSerializer);
   MessageRouter messageRouter = new MessageRouterImpl();
   var peerRegistry = new PeerRegistryImpl(pofSerializer);
   var networkReceiver = new NetworkReceiverImpl(endpoint, networkContext, pofSerializer, messageRouter, messageAck
   networkReceiver.Initialize();

   return new CourierClientFacadeImpl(endpoint, messageSender, messageRouter, peerRegistry);
}
```

Problem: Wiring up Code is Annoying

- Traditional OOP - Our custom-code directly invokes reusable dependencies.
- Inversion of Control - We pass custom dependencies into reusable objects.



```
private static DaemonServiceImpl CreateDaemonCore(IEggParameters parameters, out
    keepalive = new List<object>();
    keepalive.Add(parameters);

    // construct libwarty dependencies
    ICollectionFactory collectionFactory = new CollectionFactory();

    // construct libwarty-proxies dependencies
    IStreamFactory streamFactory = new StreamFactory();
    IFileSystemProxy fileSystemProxy = new FileSystemProxy(streamFactory);
    IThreadingFactory threadingFactory = new ThreadingFactory();
    ISynchronizationFactory synchronizationFactory = new SynchronizationFactory()
    IThreadingProxy threadingProxy = new ThreadingProxy(threadingFactory, synchro
    IDnsProxy dnsProxy = new DnsProxy();
    ITcpEndPointFactory tcpEndPointFactory = new TcpEndPointFactory(dnsProxy);
    INetworkingInternalFactory networkingInternalFactory = new NetworkingInternal
    ISocketFactory socketFactory = new SocketFactory(tcpEndPointFactory, networki
    INetworkingProxy networkingProxy = new NetworkingProxy(socketFactory, tcpEndP
    IProcessProxy processProxy = new ProcessProxy();

    // construct Castle.Core dependencies
    ProxyGenerator proxyGenerator = new ProxyGenerator();

    // construct dargon common Portable Object Format dependencies
    IPofContext pofContext = new ClientPofContext();
    IPofSerializer pofSerializer = new PofSerializer(pofContext);

    // construct libdargon.management dependencies
    ITcpEndPoint managementServerEndpoint = networkingProxy.CreateAnyEndPoint(kDa
    var managementFactory = new ManagementFactoryImpl(collectionFactory, threadin
    var localManagementServer = managementFactory.CreateServer(new ManagementServ
    keepalive.Add(localManagementServer);

    // construct root Dargon dependencies.
    var clientConfiguration = new ClientConfiguration();

    // construct system-state dependencies
    var systemState = new ClientSystemStateFactory(fileSystemProxy, clientConfigu
    localManagementServer.RegisterInstance(new ClientSystemStateMob(systemState))

    // construct libdsp dependencies
    PofStreamsFactory pofStreamsFactory = new PofStreamsFactoryImpl(threadingProx
    ServiceClientFactoryImpl serviceClientFactory = new ServiceClientFactoryImpl(

    // construct libdsp local service node
    ClusteringConfiguration clusteringConfiguration = new ClientClusteringConfigu
    ServiceClient localServiceClient = serviceClientFactory.Construct(clusteringC
    keepalive.Add(localServiceClient);

    // construct Dargon Daemon dependencies
    var core = new DaemonServiceImpl(parameters.Host, clientConfiguration);
    DaemonService daemonService = core;
    localServiceClient.RegisterService(daemonService, typeof(DaemonService));
    localManagementServer.RegisterInstance(new DaemonServiceMob(core));

    // construct miscellanious common Dargon dependencies
    TemporaryFileService temporaryFileService = new TemporaryFileServiceImpl(clie
    localServiceClient.RegisterService(temporaryFileService, typeof(TemporaryFile
    IDtpNodeFactory dtpNodeFactory = new DefaultDtpNodeFactory();

    // get the exeggutor service
    ExeggutorService exeggutorService = localServiceClient.GetService<ExeggutorSe

    // construct modification and repository dependencies
    var modificationComponentFactory = new ModificationComponentFactory(fileSyste
    ModificationLoader modificationLoader = new ModificationLoaderImpl(clientConf

    // construct process watching/injection dependencies
      IProcessInjector processInjector = new ProcessInjector();
    IProcessDiscoveryMethodFactory processDiscoveryMethodFactory = new ProcessDis
    IProcessDiscoveryMethod processDiscoveryMethod = processDiscoveryMethodFactor
    IProcessWatcher processWatcher = new ProcessWatcher(processProxy, processDisc
    TrayService trayService = new TrayServiceImpl(daemonService);
      IProcessInjectionConfiguration processInjectionConfiguration = new ProcessI
      ProcessInjectionService processInjectionService = new ProcessInjectionServi
    ProcessWatcherService processWatcherService = new ProcessWatcherServiceImpl(p
    //          ISessionFactory sessionFactory = new SessionFactory(dtpNodeFactory
    //          IInjectedModuleServiceConfiguration injectedModuleServiceConfigura
    //          InjectedModuleService injectedModuleService = new InjectedModuleSe
    //          localServiceClient.RegisterService(injectedModuleService, typeof(I
    //          localServiceClient.RegisterService(processInjectionService, typeof

    CommandFactory commandFactory = new CommandFactoryImpl();
    TrinketSpawner trinketSpawner = new TrinketSpawnerImpl(streamFactory, pofSeri

      var sectors = new SectorCollection();
      sectors.AssignSector(new SectorRange(0, 26), new FileSector("C:/dummy-files
      sectors.AssignSector(new SectorRange(28, 54), new FileSector("C:/dummy-file
      sectors.AssignSector(new SectorRange(54, 64), new FileSector("C:/dummy-file
      sectors.AssignSector(new SectorRange(65, 68), new FileSector("C:/dummy-file
      var vfm = new VirtualFile(sectors);
      var vfmSerializer = new SectorCollectionSerializer();
      using (var fs = fileSystemProxy.OpenFile("C:/dummy-files/z.vfm", FileMode.C
      using (var writer = fs.Writer) {
         vfmSerializer.Serialize(sectors, writer.__Writer);
      }

      var targetProcess = processWatcher.FindProcess(x => x.ProcessName.Contains(
      logger.Info("TARGET PID " + targetProcess.Id);
      trinketSpawner.SpawnTrinket(
         targetProcess,
         new TrinketSpawnConfigurationImpl {
            Name = "npp",
            IsFileSystemOverridingEnabled = true,
            IsFileSystemHookingEnabled = true,
            IsCommandingEnabled = true,
            CommandList = new DefaultCommandList(new [] {
               commandFactory.CreateFileRemappingCommand("C:/dummy-files/a.txt",
            })
         }
      );
      logger.Info("########################################");

    // construct additional Dargon dependencies
    IGameHandler leagueGameServiceImpl = new LeagueGameServiceImpl(clientConfigur
    IGameHandler ffxiiiGameServiceImpl = new FFXIIIGameServiceImpl(daemonService,

    return core;
}
```



```
// construct Castle.Core dependencies
ProxyGenerator proxyGenerator = new ProxyGenerator();

// construct dargon common Portable Object Format dependencies
IPofContext pofContext = new ClientPofContext();
IPofSerializer pofSerializer = new PofSerializer(pofContext);

// construct libdargon.management dependencies
ITcpEndPoint managementServerEndpoint = networkingProxy.CreateAnyEndPoint(kDaemonManagementPort);
var managementFactory = new ManagementFactoryImpl(collectionFactory, threadingProxy, networkingProxy, pofCont
var localManagementServer = managementFactory.CreateServer(new ManagementServerConfiguration(managementServer
keepalive.Add(localManagementServer);

// construct root Dargon dependencies.
var clientConfiguration = new ClientConfiguration();

// construct system-state dependencies
var systemState = new ClientSystemStateFactory(fileSystemProxy, clientConfiguration).Create();
localManagementServer.RegisterInstance(new ClientSystemStateMob(systemState));

// construct libdsp dependencies
PofStreamsFactory pofStreamsFactory = new PofStreamsFactoryImpl(threadingProxy, streamFactory, pofSerializer)
ServiceClientFactoryImpl serviceClientFactory = new ServiceClientFactoryImpl(proxyGenerator, streamFactory, c

// construct libdsp local service node
ClusteringConfiguration clusteringConfiguration = new ClientClusteringConfiguration();
ServiceClient localServiceClient = serviceClientFactory.Construct(clusteringConfiguration);
keepalive.Add(localServiceClient);
```

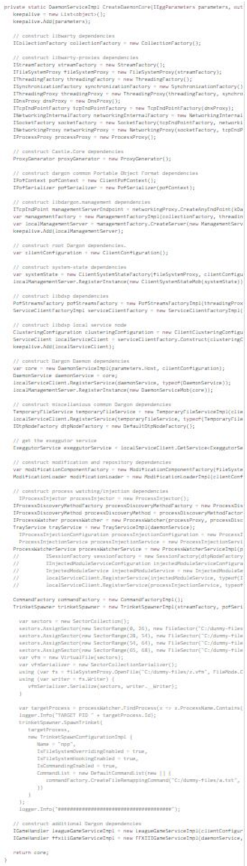
Problem: Wiring up Code is Annoying

- Traditional OOP - Our custom-code directly invokes reusable dependencies.
- Inversion of Control - We pass custom dependencies into reusable objects.
  - Annoying - Can have a TON of dependencies.
  - Have to duplicate initialization code across all startup projects.

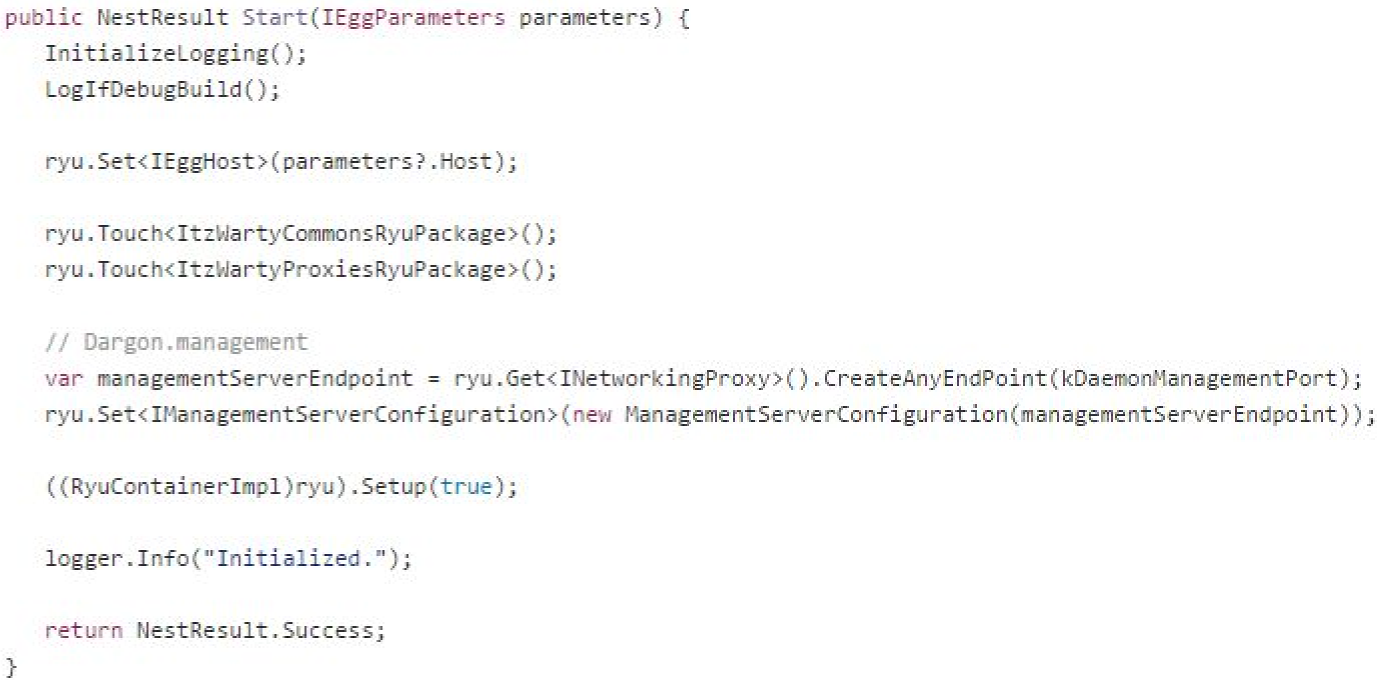
Problem: Wiring up Code is Annoying

- Traditional OOP - Our custom-code directly invokes reusable dependencies.
- Inversion of Control - We pass custom dependencies into reusable objects.
    - Annoying - Can have a TON of dependencies.
    - Have to duplicate initialization code across all startup projects.
- Solution: IoC Containers!
    - “Magically” handle instantiation of dependency tree.
        - Still works when dependencies are upgraded!
            - No need to fix new() because class has new constructor dependency.
    - Reusable across all startup projects.

IoC Containers are MAGIC!



```
private static DaemonServiceImpl CreateDaemonCore(IEggParameters parameters, out
   keepalive = new List<object>();
   keepalive.Add(parameters);

   // construct libwarty dependencies
   ICollectionFactory collectionFactory = new CollectionFactory();

   // construct libwarty-proxies dependencies
   IStreamFactory streamFactory = new StreamFactory();
   IFileSystemProxy fileSystemProxy = new FileSystemProxy(streamFactory);
   IThreadingFactory threadingFactory = new ThreadingFactory();
   ISynchronizationFactory synchronizationFactory = new SynchronizationFactory()
   IThreadingProxy threadingProxy = new ThreadingProxy(threadingFactory, synchro
   IDnsProxy dnsProxy = new DnsProxy();
   ITcpEndPointFactory tcpEndPointFactory = new TcpEndPointFactory(dnsProxy);
   INetworkingInternalFactory networkingInternalFactory = new NetworkingInternal
   ISocketFactory socketFactory = new SocketFactory(tcpEndPointFactory, networki
   INetworkingProxy networkingProxy = new NetworkingProxy(socketFactory, tcpEndP
   IProcessProxy processProxy = new ProcessProxy();

   // construct Castle.Core dependencies
   ProxyGenerator proxyGenerator = new ProxyGenerator();

   // construct dargon common Portable Object Format dependencies
   IPofContext pofContext = new ClientPofContext();
   IPofSerializer pofSerializer = new PofSerializer(pofContext);

   // construct libdargon.management dependencies
   ITcpEndPoint managementServerEndpoint = networkingProxy.CreateAnyEndPoint(kDa
   var managementFactory = new ManagementFactoryImpl(collectionFactory, threadin
   var localManagementServer = managementFactory.CreateServer(new ManagementServ
   keepalive.Add(localManagementServer);

   // construct root Dargon dependencies.
   var clientConfiguration = new ClientConfiguration();

   // construct system-state dependencies
   var systemState = new ClientSystemStateFactory(fileSystemProxy, clientConfigu
   localManagementServer.RegisterInstance(new ClientSystemStateMob(systemState))

   // construct libdsp dependencies
   PofStreamsFactory pofStreamsFactory = new PofStreamsFactoryImpl(threadingProx
   ServiceClientFactoryImpl serviceClientFactory = new ServiceClientFactoryImpl(

   // construct libdsp local service node
   ClusteringConfiguration clusteringConfiguration = new ClientClusteringConfigu
   ServiceClient localServiceClient = serviceClientFactory.Construct(clusteringC
   keepalive.Add(localServiceClient);

   // construct Dargon Daemon dependencies
   var core = new DaemonServiceImpl(parameters.Host, clientConfiguration);
   DaemonService daemonService = core;
   localServiceClient.RegisterService(daemonService, typeof(DaemonService));
   localManagementServer.RegisterInstance(new DaemonServiceMob(core));

   // construct miscellanious common Dargon dependencies
   TemporaryFileService temporaryFileService = new TemporaryFileServiceImpl(clie
   localServiceClient.RegisterService(temporaryFileService, typeof(TemporaryFile
   IDtpNodeFactory dtpNodeFactory = new DefaultDtpNodeFactory();

   // get the exeggutor service
   ExeggutorService exeggutorService = localServiceClient.GetService<ExeggutorSe

   // construct modification and repository dependencies
   var modificationComponentFactory = new ModificationComponentFactory(fileSyste
   ModificationLoader modificationLoader = new ModificationLoaderImpl(clientConf

   // construct process watching/injection dependencies
     IProcessInjector processInjector = new ProcessInjector();
   IProcessDiscoveryMethodFactory processDiscoveryMethodFactory = new ProcessDis
   IProcessDiscoveryMethod processDiscoveryMethod = processDiscoveryMethodFactor
   IProcessWatcher processWatcher = new ProcessWatcher(processProxy, processDisc
   TrayService trayService = new TrayServiceImpl(daemonService);
     IProcessInjectionConfiguration processInjectionConfiguration = new ProcessI
     ProcessInjectionService processInjectionService = new ProcessInjectionServi
   ProcessWatcherService processWatcherService = new ProcessWatcherServiceImpl(p
   //         ISessionFactory sessionFactory = new SessionFactory(dtpNodeFactory
   //         IInjectedModuleServiceConfiguration injectedModuleServiceConfigura
   //         InjectedModuleService injectedModuleService = new InjectedModuleSe
   //         localServiceClient.RegisterService(injectedModuleService, typeof(I
   //         localServiceClient.RegisterService(processInjectionService, typeof

   CommandFactory commandFactory = new CommandFactoryImpl();
   TrinketSpawner trinketSpawner = new TrinketSpawnerImpl(streamFactory, pofSeri

     var sectors = new SectorCollection();
     sectors.AssignSector(new SectorRange(0, 26), new FileSector("C:/dummy-files
     sectors.AssignSector(new SectorRange(28, 54), new FileSector("C:/dummy-file
     sectors.AssignSector(new SectorRange(54, 64), new FileSector("C:/dummy-file
     sectors.AssignSector(new SectorRange(65, 68), new FileSector("C:/dummy-file
     var vfm = new VirtualFile(sectors);
     var vfmSerializer = new SectorCollectionSerializer();
     using (var fs = fileSystemProxy.OpenFile("C:/dummy-files/x.vfm", FileMode.C
     using (var writer = fs.Writer) {
        vfmSerializer.Serialize(sectors, writer.__Writer);
     }

     var targetProcess = processWatcher.FindProcess(x => x.ProcessName.Contains(
     logger.Info("TARGET PID " + targetProcess.Id);
     trinketSpawner.SpawnTrinket(
        targetProcess,
        new TrinketSpawnConfigurationImpl {
           Name = "npp",
           IsFileSystemOverridingEnabled = true,
           IsFileSystemHookingEnabled = true,
           IsCommandingEnabled = true,
           CommandList = new DefaultCommandList(new [] {
              commandFactory.CreateFileRemappingCommand("C:/dummy-files/a.txt",
           })
        }
     );
     logger.Info("#########################################");

   // construct additional Dargon dependencies
   IGameHandler leagueGameServiceImpl = new LeagueGameServiceImpl(clientConfigur
   IGameHandler ffxiiiGameServiceImpl = new FFXIIIGameServiceImpl(daemonService,

   return core;
}
```

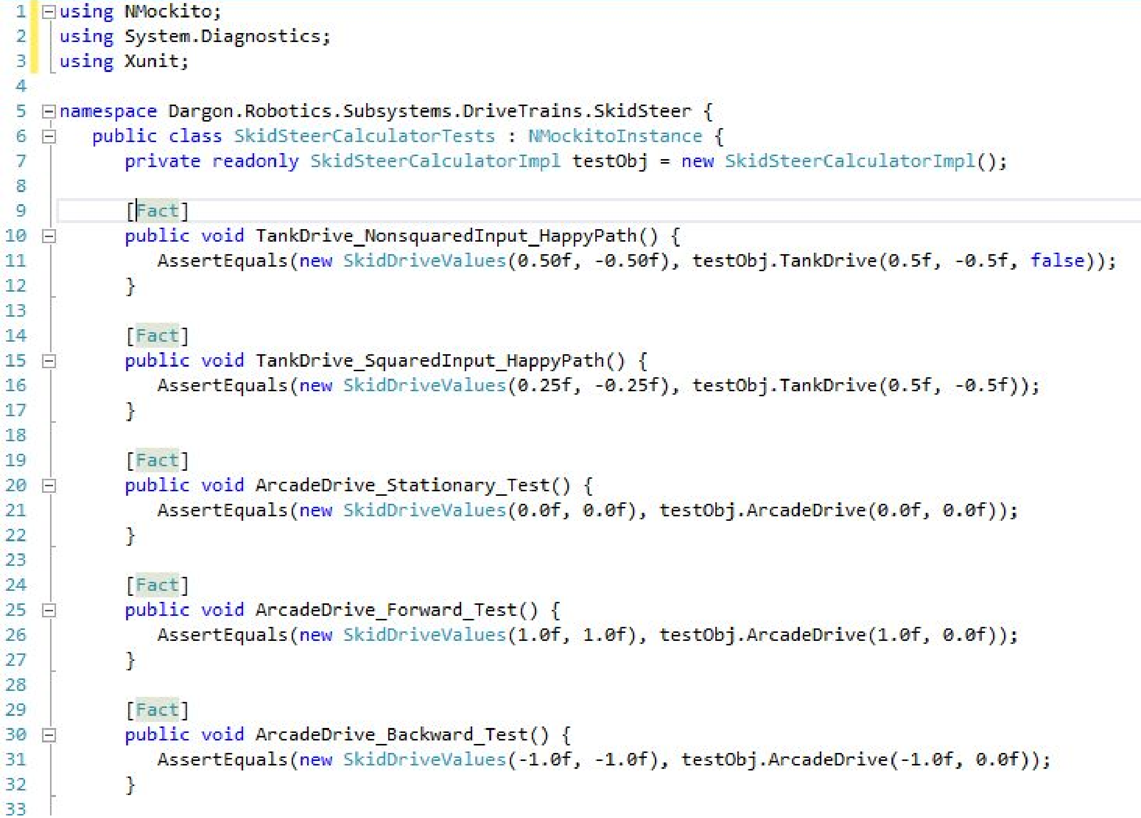


```
public NestResult Start(IEggParameters parameters) {
   InitializeLogging();
   LogIfDebugBuild();

   ryu.Set<IEggHost>(parameters?.Host);

   ryu.Touch<ItzWartyCommonsRyuPackage>();
   ryu.Touch<ItzWartyProxiesRyuPackage>();

   // Dargon.management
   var managementServerEndpoint = ryu.Get<INetworkingProxy>().CreateAnyEndPoint(kDaemonManagementPort);
   ryu.Set<IManagementServerConfiguration>(new ManagementServerConfiguration(managementServerEndpoint));

   ((RyuContainerImpl)ryu).Setup(true);

   logger.Info("Initialized.");

   return NestResult.Success;
}
```

Automated Testing

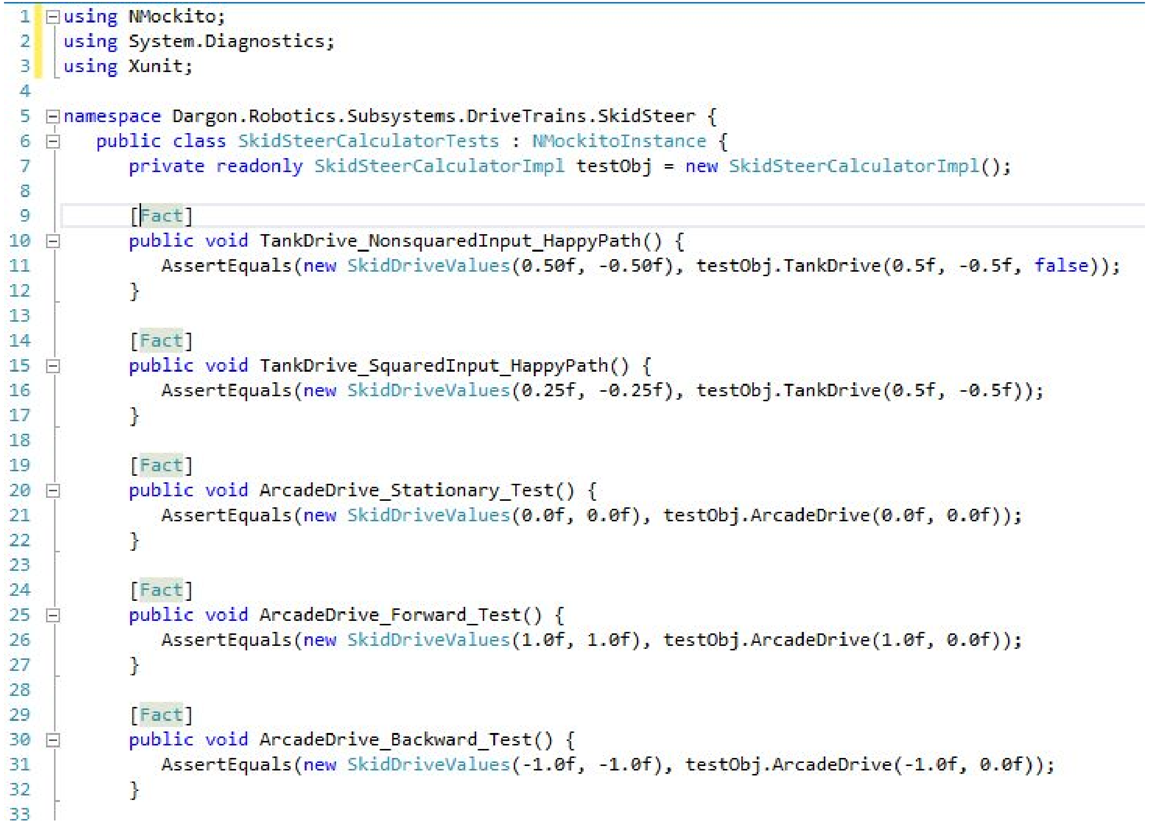
- Terminology varies.
- Term: SUT - Subject Under Test (named `testObj` in our code)
- Different ‘categories’ of tests
    - Boundary between tests might be ‘gray’ at times.
    - Unit Tests - Test interaction between SUT and dependencies (“SUT delegates to Dep”)
        - SUT should be isolated from FileSystem, Dependencies,etc.
    - Integration Tests - Tests multiple subjects in system as one “unit”
        - Example: Code path through from SUT to Dep1 to Dep2 touches Dep3
    - Functional Tests - Black Box Tests of system input/output (“Input X must yield Output Y”)
        - Great for telling you “Hey, it doesn’t work”.
        - Bad at telling you “Why is it not working?”.
    - Acceptance Testing - E.g. “Is it fast enough” or “Does it actually work?”

Example of a Unit Test



```
using NMockito;
using System.Diagnostics;
using Xunit;

namespace Dargon.Robotics.Subsystems.DriveTrains.SkidSteer {
   public class SkidSteerCalculatorTests : NMockitoInstance {
      private readonly SkidSteerCalculatorImpl testObj = new SkidSteerCalculatorImpl();

      [Fact]
      public void TankDrive_NonsquaredInput_HappyPath() {
         AssertEquals(new SkidDriveValues(0.50f, -0.50f), testObj.TankDrive(0.5f, -0.5f, false));
      }

      [Fact]
      public void TankDrive_SquaredInput_HappyPath() {
         AssertEquals(new SkidDriveValues(0.25f, -0.25f), testObj.TankDrive(0.5f, -0.5f));
      }

      [Fact]
      public void ArcadeDrive_Stationary_Test() {
         AssertEquals(new SkidDriveValues(0.0f, 0.0f), testObj.ArcadeDrive(0.0f, 0.0f));
      }

      [Fact]
      public void ArcadeDrive_Forward_Test() {
         AssertEquals(new SkidDriveValues(1.0f, 1.0f), testObj.ArcadeDrive(1.0f, 0.0f));
      }

      [Fact]
      public void ArcadeDrive_Backward_Test() {
         AssertEquals(new SkidDriveValues(-1.0f, -1.0f), testObj.ArcadeDrive(-1.0f, 0.0f));
      }

```

Example of a Unit Test



```
using NMockito;
using System.Diagnostics;
using Xunit;

namespace Dargon.Robotics.Subsystems.DriveTrains.SkidSteer {
   public class SkidSteerCalculatorTests : NMockitoInstance {
      private readonly SkidSteerCalculatorImpl testObj = new SkidSteerCalculatorImpl();

      [Fact]
      public void TankDrive_NonsquaredInput_HappyPath() {
         AssertEquals(new SkidDriveValues(0.50f, -0.50f), testObj.TankDrive(0.5f, -0.5f, false));
      }

      [Fact]
      public void TankDrive_SquaredInput_HappyPath() {
         AssertEquals(new SkidDriveValues(0.25f, -0.25f), testObj.TankDrive(0.5f, -0.5f));
      }

      [Fact]
      public void ArcadeDrive_Stationary_Test() {
         AssertEquals(new SkidDriveValues(0.0f, 0.0f), testObj.ArcadeDrive(0.0f, 0.0f));
      }

      [Fact]
      public void ArcadeDrive_Forward_Test() {
         AssertEquals(new SkidDriveValues(1.0f, 1.0f), testObj.ArcadeDrive(1.0f, 0.0f));
      }

      [Fact]
      public void ArcadeDrive_Backward_Test() {
         AssertEquals(new SkidDriveValues(-1.0f, -1.0f), testObj.ArcadeDrive(-1.0f, 0.0f));
      }

```

Example of a Unit Test with Mocking

https://github.com/the-dargon-project/NMockito/blob/master/NMockito.Tests/ExampleTest.cs