Chapter 1

Robot Code

Robot.java

```java
package frc.robot;

import edu.wpi.first.wpilibj.TimedRobot;
import frc.robot.Mode.Autonomous;
import frc.robot.Mode.Disabled;
import frc.robot.Mode.Simulation;
import frc.robot.Mode.Teleop;
import frc.robot.Mode.Test;
import frc.robot.Mode.Onabot;

public class Robot extends TimedRobot {
  @Override public void robotInit          () { Onabot     .Initialize(); }
  @Override public void robotPeriodic      () { Onabot     .Periodic();   }

  @Override public void autonomousInit     () { Autonomous .Initialize(); }
  @Override public void autonomousPeriodic () { Autonomous .Periodic();   }

  @Override public void disabledInit       () { Disabled   .Initialize(); }
  @Override public void disabledPeriodic   () { Disabled   .Periodic();   }

  @Override public void teleopInit         () { Teleop     .Initialize(); }
  @Override public void teleopPeriodic     () { Teleop     .Periodic();   }

  @Override public void testInit           () { Test       .Initialize(); }
  @Override public void testPeriodic       () { Test       .Periodic();   }

  @Override public void simulationInit     () { Simulation .Initialize(); }
  @Override public void simulationPeriodic () { Simulation .Periodic();   }
}
```

Onabot.java

```java
package frc.robot.Mode;

import frc.robot.Hardware.Driver;
import frc.robot.Hardware.Elevator;
import frc.robot.Hardware.Navigation;
import frc.robot.Hardware.Swerve;

public class Onabot {

    public static void Initialize () {
        Driver     .Initialize();
        Elevator   .Initialize();
        Navigation .Initialize();
        Swerve     .Initialize();

        // Sonar      .Initialize();
        // Vision     .Initialize();
    }

    public static void Periodic () {
        Driver     .Display();
        Elevator   .Display();
        Navigation .Display();
        Swerve     .Display();

        // Sonar      .Display();
        // Vision     .Display();
    }

}
```

Autonomous.java

```
package frc.robot.Mode;

import edu.wpi.first.wpilibj.smartdashboard.SendableChooser;
import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
import frc.robot.Hardware.Autopilot;
import frc.robot.Hardware.Elevator;
import frc.robot.Hardware.Stage;
import frc.robot.Hardware.Swerve;
import frc.robot.Hardware.Track;

public class Autonomous {

    public static final String kDefault = "Nothing";
    public static final String kPath01  = "Path-01";
    public static final String kPath02  = "Path-02";
    public static final String kPath03  = "Path-03";
    public static final SendableChooser<String> chooser = new SendableChooser<>();

    public static void Initialize () {
        chooser.setDefaultOption("Nothing", kDefault );
        chooser.setDefaultOption("Path␣01", kPath01  );
        chooser.setDefaultOption("Path␣02", kPath02  );
        chooser.setDefaultOption("Path␣03", kPath03  );
        SmartDashboard.putData  ("PATH",    chooser  );

        Stage.Initialize();
    }

    public static void Periodic () {
        Stage.Begin();

        switch ( chooser.getSelected() ) {
            case "Default" : Track.Track_00(); break;
            case "Path-01" : Track.Track_01(); break;
            case "Path-02" : Track.Track_02(); break;
            case "Path-03" : Track.Track_03(); break;
        }

        Stage.Display();

        Elevator.Periodic();
        Swerve.UpdateFieldRelative( Autopilot.vx, Autopilot.vy, Autopilot.vt );

        // Example chassis speeds: 1 meter per second forward, 3 meters
        // per second to the left, and rotation at 1.5 radians per second
        // counterclockwise.                        F    L   CCW
        // ChassisSpeeds speeds = new ChassisSpeeds(1.0, 3.0, 1.5);

        // THESE NEED TO BE SET BY THE AUTONOMOUS MODE

        // double curPitch  = Navigation.GetPitch();
        // SmartDashboard.putNumber("Robot-Pitch", curPitch);
        // SmartDashboard.putNumber("Robot-Stage", stage );

        // double
        //      vx = 0.00,
        //      vy = 0.00,
        //      vt = 0.00;

        // if ( stage == 0 ) {
        //      vx = 0.08;
        //      if ( curPitch > 0 ) { stage++; };
        // }

        // if ( stage == 1 ) {
```

```
        //       vx = 0;
        // }

        // Navigation.Periodic();
        // double curAng = Navigation.GetDirection();
        // double target = 0;

        // double diff = curAng − target;

        // SmartDashboard.putNumber(”Robot–DIFF”, diff);

        // // SMALLEST ANGLE TO SWIVEL: −180 to 180
        // double minTurn = ( diff + 180 ) % 360 − 180;
        //       double turnMag = Math.abs    ( minTurn );
        //       double turnDir = Math.signum( minTurn );

        // // MINIMIZE WHEEL SWIVEL: +120 becomes −60
        // if ( turnMag > 0 ) {
        //       // turnMag  = 180 − minTurn; // Turn smaller angle
        //       turnDir ∗= −1;             // and reverse swivel
        // }

        // // DETERMINE POWER USING PSEUDO PID CONTROLLER
        // if       ( turnMag > 20 ) { vt = 0.15; }
        // else if ( turnMag > 10 ) { vt = 0.10; }
        // else if ( turnMag >  3 ) { vt = 0.06; }
        // else if ( turnMag >  2 ) { vt = 0.00; }
        // else if ( turnMag >  1 ) { vt = 0.00; }
        // else                      { vt = 0.00; }

        // vt ∗= turnDir;

        // SET MOTOR CONTROLLERS
        // SteerMotor.set( TalonFXControlMode.PercentOutput, PID.calculate( SP, PV ) );

        // double diff = ( cur   ) % 360 − 180;

    }

}
```

Teleop.java

```java
package frc.robot.Mode;

import edu.wpi.first.wpilibj.Joystick;
import frc.robot.Hardware.Driver;
import frc.robot.Hardware.Elevator;
import frc.robot.Hardware.Settings;

public class Teleop {

    public static Joystick DriveStick;
    public static Joystick ManipStick;

    public static double Xratio;
    public static double Yratio;
    public static double Tratio;

    public static void Initialize () {
        DriveStick = new Joystick( Settings.DriveStickID );
        ManipStick = new Joystick( Settings.ManipStickID );
    }

    public static void Periodic () {
        Xratio = -DriveStick.getY();
        Yratio = -DriveStick.getX();
        Tratio = -DriveStick.getTwist();

        Driver.Periodic();
        Elevator.Periodic();
    }

    public static void Display () {
    }
}
```

Autopilot.java

The Autopilot methods are used in Autonomous mode to set the chassis speed variable found in this class. Values are sent to motor controllers in Autonomous.Periofic().

```java
package frc.robot.Hardware;

public class Autopilot {

    public static double LastHeading = 0;

    public static double vx = 0;
    public static double vy = 0;
    public static double vt = 0;

//
// HeadingDiff is a simple method that calculates the angle difference
// between the current and desired heading. This can be used anywhere.
//
    public static double HeadingDiff ( double SP ) {

        // CALCULATE TURN VALUES
          double PV = Navigation.GetDirection(); // Current state (Initial)
                SP = ( SP + 360 ) % 360;         // Ensure SP is between 0 and 360
                double diff = -( SP - PV );      // Why is this negated? Should setInverted have been used?

          // SMALLEST ANGLE TO SWIVEL: -180 to 180
          double minTurn = ( diff + 180 ) % 360 - 180;
        return minTurn;
    }

//
// This is a simple method for driving somewhat straight without using
// a gyroscope. There may be situations where it is good enough.
//
    public static void DriveSortaStraight ( double Vx, double Vy ) {
        vx = Vx; vy = Vy; vt = 0;
    }

    public static void DriveStraight ( double Vx, double Vy ) {
        vx = Vx; vy = Vy; vt = 0;
    }

//
//
//
    public static void DriveNorth ( double Speed ) {
        vx = +Speed; vy = 0; vt = 0;
    }

    public static void DriveSouth ( double Speed ) {
        vx = -Speed; vy = 0; vt = 0;
    }

    public static void DriveWest ( double Speed ) {
        vx = 0; vy = +Speed; vt = 0;
    }

    public static void DriveEast ( double Speed ) {
        vx = Speed; vy = -Speed; vt = 0;
    }

//
//
//
    public static void DriveNorthWest ( double Speed ) {
        double radians = Math.toRadians( 45 );
        double speed   = Speed * Math.cos( radians );
```

```
      vx = +speed; vy = +speed; vt = 0;
   }

   public static void DriveNorthEast ( double Speed ) {
      double radians = Math.toRadians( 45 );
      double speed   = Speed * Math.cos( radians );
      vx = +speed; vy = -speed; vt = 0;
   }

   public static void DriveSouthWest ( double Speed ) {
      double radians = Math.toRadians( 45 );
      double speed   = Speed * Math.cos( radians );
      vx = -speed; vy = +speed; vt = 0;
   }

   public static void DriveSouthEast ( double Speed ) {
      double radians = Math.toRadians( 45 );
      double speed   = Speed * Math.cos( radians );
      vx = -speed; vy = -speed; vt = 0;
   }

//
//
//
   public static void SetWheelsToHeading ( double Angle) {
      Swerve.SetWheelsToHeading( Angle );
   }

//
// TurnToHeading sets the turn power variable in Autonomous mode to reach
// the desired heading using the shortest wheek swivel.
//
   public static void TurnToHeading ( double NewHeading ) {
      double minTurn = HeadingDiff( NewHeading );
      double turnMag = Math.abs    ( minTurn );
      double turnDir = Math.signum( minTurn );

        // MINIMIZE WHEEL SWIVEL: +120 becomes -60
        if ( turnMag > 0 ) {
            turnMag  = 180 - minTurn; // Turn smaller angle
            turnDir *= -1;            // and reverse swivel
        }

        // DETERMINE POWER USING PSEUDO PID CONTROLLER
        if      ( turnMag > 20 ) { vt = 0.15; }
        else if ( turnMag > 10 ) { vt = 0.10; }
        else if ( turnMag >  3 ) { vt = 0.06; }
        else if ( turnMag >  2 ) { vt = 0.00; }
        else if ( turnMag >  1 ) { vt = 0.00; }
        else                     { vt = 0.00; }

      LastHeading = NewHeading;
      vx = 0; vy = 0; vt *= turnDir;
   }

//
// Stop sets the robot speed vector to zero. This is useful only in Autonomous
// mode. It should not be used elsewhere.
//
   public static void Stop () {
      vx = 0; vy = 0; vt = 0;
   }

//
// These methods rotate the robot at a constant counter-clockwise speed and
// clockwise speed respectively. This is only useful in Autonomous mode.
//
   public static void TurnLeftAtSpeed ( double Speed ) {
      vx = 0; vy = 0; vt = +Speed;
   }

   public static void TurnRightAtSpeed ( double Speed ) {
      vx = 0; vy = 0; vt = -Speed;
   }

   // public static void AdjustTurnSpeed( double Speed ) {
   //    // double error = Speed - Navigation.GetTurnSpeed();
   //    // LastPowerT  += error * 0.0001;
   // }

   // public static void DriveStraight ( double Speed, double Heading ) {

   // }

}
```

Driver.java

```java
package frc.robot.Hardware;

import edu.wpi.first.wpilibj.smartdashboard.SendableChooser;
import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
import frc.robot.Driver.Aubrey;
import frc.robot.Driver.Default;
import frc.robot.Driver.Nate;
import frc.robot.Driver.Steensma;

public class Driver {

    public static final String kDefault  = "Default";
    public static final String kAubrey   = "Aubrey";
    public static final String kNate     = "Nate";
    public static final String kSteensma = "Steensma";
    public static final SendableChooser<String> chooser = new SendableChooser<>();

    public static void Initialize () {
        chooser.setDefaultOption("Default",  kDefault  );
        chooser.addOption       ("Aubrey",   kAubrey   );
        chooser.addOption       ("Nate",     kNate     );
        chooser.addOption       ("Steensma", kSteensma );
        SmartDashboard.putData  ("DRIVER",   chooser   );
    }

    public static void Periodic () {
        switch ( chooser.getSelected() ) {
            case "Default" : Default  .Periodic();
            case "Aubrey"  : Aubrey   .Periodic();
            case "Nate"    : Nate     .Periodic();
            case "Steensma": Steensma .Periodic();
        }
    }

    public static void Display () {
    }

}
```

Elevator.java

```java
package frc.robot.Hardware;

public class Elevator {

    public static void Initialize () {
        ElevArm     .Initialize();
        ElevClaw    .Initialize();
        ElevLift    .Initialize();
        ElevWrist   .Initialize();
    }

    public static void Periodic () {
        ElevArm     .Periodic();
        ElevClaw    .Periodic();
        ElevLift    .Periodic();
        ElevWrist   .Periodic();
    }

    public static void Display () {
        ElevArm     .Display();
        ElevClaw    .Display();
        ElevLift    .Display();
        ElevWrist   .Display();
    }

//
//
//
    public static void Reset () {
        ElevArm     .Reset();
        ElevClaw    .Reset();
        ElevLift    .Reset();
        ElevWrist   .Reset();
    }

    public static void Set ( double A, double C, double L, double W ) {
        ElevArm     .SetPosition( A );
        ElevClaw    .SetPosition( C );
        ElevLift    .SetPosition( L );
        ElevWrist   .SetPosition( W );
    }

}
```

ElevArm.java

```java
package frc.robot.Hardware;

import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;

public class ElevArm {

    public static double target_position;

//
//
//
    public static void Initialize () {

    }

    public static void Periodic () {
        // Some sort of controller to find the position
        // difference and set the motor ratio. Might need
        // a PID controller to hold position.
    }

    public static void Display () {
        SmartDashboard.putNumber("Elevator-Arm␣Pos", GetPosition()    );
        SmartDashboard.putNumber("Elevator-Arm␣Tar", target_position );
    }

//
//
//
    public static double GetPosition () {
        return 0;
    }

    public static void SetPosition ( double pos ) {
        target_position = pos;
    }

//
//
//
    public static void Reset () {
        Retract();
    }

    public static void Extend () {

    }

    public static void Retract () {

    }

}
```

ElevClaw.java

```java
package frc.robot.Hardware;

import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;

public class ElevClaw {

    public static double target_position;

//
//
//
    public static void Initialize () {

    }

    public static void Periodic () {
        // Some sort of controller to find the position
        // difference and set the motor ratio. Might need
        // a PID controller to hold position.
    }

    public static void Display () {
        SmartDashboard.putNumber("Elevator-Claw␣Pos", GetPosition()    );
        SmartDashboard.putNumber("Elevator-Claw␣Tar", target_position );
    }

//
//
//
    public static void Reset () {
        Open();
    }

    public static void Grab () {

    }

    public static void Open () {

    }

//
//
//
    public static double GetPosition () {
        return 0;
    }

    public static void SetPosition ( double pos ) {
        target_position = pos;
    }

}
```

ElevLift.java

```java
package frc.robot.Hardware;

import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;

public class ElevWrist {

    public static double target_position;

//
//
//
    public static void Initialize () {

    }

    public static void Periodic () {
        // Some sort of controller to find the position
        // difference and set the motor ratio. Might need
        // a PID controller to hold position.
    }

    public static void Display () {
        SmartDashboard.putNumber("Elevator-Wrist␣Pos", GetPosition()    );
        SmartDashboard.putNumber("Elevator-Wrist␣Tar", target_position );
    }

//
//
//
    public static double GetPosition () {
        return 0;
    }

    public static void SetPosition ( double pos ) {
        target_position = pos;
    }

//
//
//
    public static void Reset () {
        Bend();
    }

    public static void Bend () {

    }

    public static void Straighten () {

    }

}
```

ElevWrist.java

```
package frc.robot.Hardware;

import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;

public class ElevWrist {

    public static double target_position;

//
//
//
    public static void Initialize () {

    }

    public static void Periodic () {
        // Some sort of controller to find the position
        // difference and set the motor ratio. Might need
        // a PID controller to hold position.
    }

    public static void Display () {
        SmartDashboard.putNumber("Elevator-Wrist␣Pos", GetPosition()     );
        SmartDashboard.putNumber("Elevator-Wrist␣Tar", target_position );
    }

//
//
//
    public static double GetPosition () {
        return 0;
    }

    public static void SetPosition ( double pos ) {
        target_position = pos;
    }

//
//
//
    public static void Reset () {
        Bend();
    }

    public static void Bend () {

    }

    public static void Straighten () {

    }

}
```

EncTalonFX.java

```
package frc.robot.Hardware;

import com.ctre.phoenix.sensors.CANCoder;

public class EncTalonFX {

    public CANCoder FalconEncoder;
    public final static  int       kUnitsPerRevolution = 2048;

    public EncTalonFX ( int CanBusID ) {
        FalconEncoder = new CANCoder( CanBusID );
    }

}
```

Module.java

```
package frc.robot.Hardware;

import com.ctre.phoenix.motorcontrol.NeutralMode;
import com.ctre.phoenix.motorcontrol.can.WPI_TalonFX;
import edu.wpi.first.math.kinematics.SwerveModuleState;

public class Module {

    EncTalonFX   SteerEncoder;
    int          ModuleNumber;
    String       ModuleName;
    WPI_TalonFX  DriveMotor;
    WPI_TalonFX  SteerMotor;

    // PIDController PID;

    public Module ( String ModuleName, int ModuleNumber ) {

        // REMEMBER VALUES
        this.ModuleName   = ModuleName;
        this.ModuleNumber = ModuleNumber;

        // ID'S FOLLOW A PATTERN BASED ON MODULE NUMBERS
        int DriveMotorID = ModuleNumber *2 -1;
        int SteerMotorID = ModuleNumber *2 -0;

        // CONFIGURE MOTOR THROUGH SOFTWARE
        // TalonFXConfiguration config = new TalonFXConfiguration();
        // config.supplyCurrLimit.enable                  = false;
        // config.supplyCurrLimit.currentLimit            = 30.0;
        // config.supplyCurrLimit.triggerThresholdCurrent = 30.0;
        // config.supplyCurrLimit.triggerThresholdTime    = 1.5;

        // DEFINE AND CONFIGURE DRIVE MOTOR
        DriveMotor = new WPI_TalonFX ( DriveMotorID );
        DriveMotor.setNeutralMode( NeutralMode.Brake );
        // driveMotor.configAllSettings ( config );

        // DEFINE AND CONFIGURE STEER MOTOR
        SteerMotor = new WPI_TalonFX ( SteerMotorID );
        SteerMotor.setNeutralMode( NeutralMode.Brake );
        // steerMotor.configAllSettings ( config );

        // DEFINE STEER ENCODER
        SteerEncoder = new EncTalonFX ( ModuleNumber );
        // SteerEncoder.FalconEncoder.setPosition( 0 );

        // PID CONTROLLER
        // PID = new PIDController( 0.1, 0, 0 );
        // PID.enableContinuousInput( 0, 360 );
        // PID.setIntegratorRange( -0.5, 0 );
        // PID.setTolerance( 5, 10 );
    }

    public void Display () {

        // DRIVE MOTOR
        // double DrivePercent = DriveMotor.getMotorOutputPercent();
        // SmartDashboard.putNumber( ModuleName + "_Drive Percent", DrivePercent );

        // STEER MOTOR
        // double SteerPercent = SteerMotor.getMotorOutputPercent();
        // SmartDashboard.putNumber( ModuleName + "_Steer Percent", SteerPercent );

        // STEER ENCODER
```

```java
        // SmartDashboard.putNumber(ModuleName + " DIFF", diff );
        // SmartDashboard.putNumber(ModuleName + " SMAL", s );
    }

    public void ResetDriveEncoder () {
        // DriveMotor
    }

    public double GetDirection () {
        return SteerEncoder.FalconEncoder.getAbsolutePosition();
    }

    public void Update( SwerveModuleState state ) {

        // CALCULATE DRIVE VALUES
        double DriveRatio = state.speedMetersPerSecond;
        double reverse    = 1;

        // CALCULATE TURN VALUES
        double SP = state.angle.getDegrees(); // Desired state (Final)
        double PV = GetDirection();           // Current state (Initial)
            SP = ( SP + 360 ) % 360;          // Ensure SP is between 0 and 360
            double diff = -( SP - PV );       // Why is this negated? Should setInverted have been used?

        // SMALLEST ANGLE TO SWIVEL: -180 to 180
        double minTurn = ( diff + 180 ) % 360 - 180;
            double turnMag = Math.abs   ( minTurn );
            double turnDir = Math.signum( minTurn );

        // MINIMIZE WHEEL SWIVEL: +120 becomes -60
        if ( turnMag > 90 ) {
            turnMag  = 180 - minTurn; // Turn smaller angle
            turnDir *= -1;            // and reverse swivel
            reverse *= -1;            // and reverse drive.
        }

        // DETERMINE POWER USING PSEUDO PID CONTROLLER
        double SteerRatio = 0;
        if      ( turnMag > 20 ) { SteerRatio = 0.20; }
        else if ( turnMag > 10 ) { SteerRatio = 0.10; }
        else if ( turnMag >  5 ) { SteerRatio = 0.06; }
        else if ( turnMag >  3 ) { SteerRatio = 0.04; }
        else if ( turnMag >  2 ) { SteerRatio = 0.00; }
        else if ( turnMag >  1 ) { SteerRatio = 0.00; }
        else                     { SteerRatio = 0.00; }

        // SET MOTOR CONTROLLERS
        DriveMotor.setVoltage( DriveRatio * 10 * reverse );
        SteerMotor.setVoltage( SteerRatio * 10 * turnDir );
        // SteerMotor.set( TalonFXControlMode.PercentOutput, PID.calculate( SP, PV ) );

        // SmartDashboard.putNumber(ModuleName + " DIFF", diff );
        // SmartDashboard.putNumber(ModuleName + " SMAL", s );
    }

    public void SetToHeading ( double Angle ) {

        // CALCULATE TURN VALUES
        double SP = Angle;                // Desired state (Final)
        double PV = GetDirection();       // Current state (Initial)
            SP = ( SP + 360 ) % 360;      // Ensure SP is between 0 and 360
            double diff = -( SP - PV );   // Why is this negated? Should setInverted have been used?

        // DETERMINE POWER USING PSEUDO PID CONTROLLER
        double SteerRatio = 0;
        if      ( diff > 40 ) { SteerRatio = 0.50; }
        else if ( diff > 20 ) { SteerRatio = 0.20; }
        else if ( diff > 10 ) { SteerRatio = 0.10; }
        else if ( diff >  5 ) { SteerRatio = 0.06; }
        else if ( diff >  3 ) { SteerRatio = 0.04; }
        else if ( diff >  2 ) { SteerRatio = 0.00; }
        else if ( diff >  1 ) { SteerRatio = 0.00; }
        else                  { SteerRatio = 0.00; }

        SteerMotor.setVoltage( SteerRatio * 10 );
    }

}
```

Navigation.java

```
package frc.robot.Hardware;

import com.kauailabs.navx.frc.AHRS;
import edu.wpi.first.wpilibj.SPI;
import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;

public class Navigation {

    public static AHRS NavX;

    public static void Initialize () {
        NavX = new AHRS( SPI.Port.kMXP );
        NavX.calibrate();
        Reset();
    }

    public static void Periodic () {}

    public static void Display() {
        SmartDashboard.putNumber( ”Nav–Yaw”,     GetYaw()    );
        SmartDashboard.putNumber( ”Nav–Pitch”,   GetPitch()  );
        SmartDashboard.putNumber( ”Nav–Roll”,    GetRoll()   );
    }

//
// SUPPORT METHODS
//
    public static void   Reset     () { NavX.reset(); }

    public static double GetPitch  () { return  NavX.getPitch(); } // Forward tilt  : – is up
    public static double GetRoll   () { return  NavX.getRoll();  } // Side–to–side  : + is ?
    public static double GetYaw    () { return –NavX.getYaw();   } // Twist         : + is CCW

    public static double GetDirection () { return GetYaw(); }
}
```

Settings.java

```java
package frc.robot.Hardware;

public class Settings {

    // CONTROLLER PORTS
    public static int
        DriveStickID = 0,
        ManipStickID = 1;

    // MAXIMUM MODULE SPEEDS
    public static double
        MAX_DRIVE_RATIO = 0.20;

    // MODULE ASSIGNMENTS
    public static int
        FL_moduleNumber = 1,
        FR_moduleNumber = 4,
        RL_moduleNumber = 5,
        RR_moduleNumber = 3;

    // MODULE LOCATIONS
    public static double
        FLx =  1, FLy =  1,
        FRx =  1, FRy = -1,
        RLx = -1, RLy =  1,
        RRx = -1, RRy = -1;

    // CLICKS PER FOOT
    public static double
        IN_PER_CLICK = ( Math.PI * 4 ) / EncTalonFX.kUnitsPerRevolution;

}
```

Stage.java

```
package frc.robot.Hardware;

import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;

public class Stage {

    public static double   AutonStartTime;
    public static double   AutonFinalTime;
    public static double   StageStartTime;

    public static int      Number;
    public static boolean  ReadyToAdvance;
    public static double[] StageDistance = new double[10];
    public static double[] StageTime     = new double[10];

    public static double   NegTilt = 0;
    public static double   PosTilt = 0;


    public static void Initialize () {
        AutonStartTime = System.currentTimeMillis();
        StageStartTime = AutonStartTime;
        Number = 0;
    }

    public static void Display () {
        SmartDashboard.putNumber(”Robot−Stage␣Number”,   Number             );
        SmartDashboard.putNumber(”Robot−Stage␣Distance”, GetDistance()      );
        SmartDashboard.putNumber(”Robot−Stage␣Time”,     GetStageTime()     );
        SmartDashboard.putNumber(”Robot−Auton␣Time”,     GetAutonDuration() );
    }

//
// The Next method advances to the next stage after storing Stage
// information. The Last method stops
//
    public static void Begin () {
        Autopilot.Stop();
        ReadyToAdvance = true;
    }

    public static void Next () {
        if ( ReadyToAdvance == true ) {
            StageDistance[Number] = GetDistance();
            StageTime    [Number] = GetStageTime();
            ResetOdometer();
            StageStartTime = System.currentTimeMillis();
            Number++;
        }
    }

    public static void Last () {
        AutonFinalTime = System.currentTimeMillis();
        ReadyToAdvance = false;
    }

    public static void Fail () {
        AutonFinalTime = System.currentTimeMillis();
        ReadyToAdvance = false;
        Number         = 100;
    }

//
// Get...Time methods are is useful in auton mode to determine the amount
// of time that the current stage or the entire auton process has been
// executing.
//
   public static double GetAutonDuration () {
      return ( System.currentTimeMillis() - AutonStartTime ) / 1000.0;
   }

   public static double GetStageTime () {
      return ( System.currentTimeMillis() - StageStartTime ) / 1000.0;
   }

   public static void WaitForDuration ( double Duration ) {
      if ( GetStageTime() < Duration ) {
         ReadyToAdvance = false;
      }
   }

//
//
//
   public static double GetDistance () {
      double FL = Swerve.FL_module.DriveMotor.getSelectedSensorPosition();
      double FR = Swerve.FL_module.DriveMotor.getSelectedSensorPosition();
      double RL = Swerve.FL_module.DriveMotor.getSelectedSensorPosition();
      double RR = Swerve.FL_module.DriveMotor.getSelectedSensorPosition();

      // ABS SINCE SOME WHEELS GOING BACKWARD
      FL = Math.abs( FL );
      FR = Math.abs( FR );
      RL = Math.abs( RL );
      RR = Math.abs( RR );

      // TAKE AN AVERAGE FOR SIMPLICITY
      return ( FL + FR + RL + RR ) * Settings.IN_PER_CLICK / 4;
   }

   public static void ResetOdometer () {
      Swerve.FL_module.DriveMotor.setSelectedSensorPosition( 0 );
      Swerve.FR_module.DriveMotor.setSelectedSensorPosition( 0 );
      Swerve.RL_module.DriveMotor.setSelectedSensorPosition( 0 );
      Swerve.RR_module.DriveMotor.setSelectedSensorPosition( 0 );
   }

   public static void WaitForDistance ( double Distance ) {
      if ( GetDistance() < Distance ) {
         ReadyToAdvance = false;
      }
   }

//
//
//
   public static void WaitForHeading ( double Heading, double Tolerance ) {
      double diff = Autopilot.HeadingDiff( Heading );
      if ( Math.abs( diff ) < Tolerance ) {
         ReadyToAdvance = false;
      }
   }

//
// Second draft of code to be used in auton. Drive forward until we notice an incline.
// At that point we advance stages and continue to drive forward until we notice a
// balanced condition. It would be good to also have a maximum distance travelled for
// each stage and fail if the condition is not met.
//
   public static void WaitForBalance ( double Tolerance ) {
      double pitch = Navigation.GetPitch();
      if ( Math.abs( pitch ) > Tolerance ) {
         ReadyToAdvance = false;
```

```
        }
    }

    public static void WaitForIncline ( double Angle ) {
        double pitch = Navigation.GetPitch();
        if ( Math.abs( pitch ) < Angle ) {
            ReadyToAdvance = false;
        }
    }

//
//
//
    // public static void WaitForWheelAlignment ( double Angle ) {

    // }
    // public static void WaitForHeading ( double Heading, double Tolerance ) {

    // }

// public static boolean WaitForHeading( double targetHeading, double tolerance ) {
//      if ( Math.abs(Navigation.GetDelta(targetHeading)) < tolerance ) { return true; }
// }
//
//
// public static boolean WaitForTarget( double tolerance ) {
//      if ( Drivetrain.TargetMin<-tolerance || Drivetrain.TargetMax>tolerance ) {
//          StillWorking = true;
//          return true;
//      }
//      else {
//          return false;
//      }
//
// }


}
```

Swerve.java

```
package frc.robot.Hardware;

import edu.wpi.first.math.geometry.Rotation2d;
import edu.wpi.first.math.geometry.Translation2d;
import edu.wpi.first.math.kinematics.ChassisSpeeds;
import edu.wpi.first.math.kinematics.SwerveDriveKinematics;
import edu.wpi.first.math.kinematics.SwerveModuleState;
import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;

public class Swerve {

    // CHASSIS SPEEDS
    public static ChassisSpeeds
        RobotSpeed;

    // MODULE DEFINITIONS
    public static Module
        FL_module,
        FR_module,
        RL_module,
        RR_module;

    // TRANSLATION OBJECTS
    static Translation2d
        FL_Trans2d,
        FR_Trans2d,
        RL_Trans2d,
        RR_Trans2d;

    // KINEMATICS OBJECT
    public static SwerveDriveKinematics
        Kinematics;

    // ODOMETRY OBJECT
    // public static SwerveDriveOdometry
    //     Odometry;

    // INITIALIZE
    public static void Initialize () {

        // CHASSIS SPEEDS
        RobotSpeed = new ChassisSpeeds( 0, 0, 0 );

        // MODULE DEFINITIONS
        FL_module = new Module( ”FL”, Settings.FL_moduleNumber );
        FR_module = new Module( ”FR”, Settings.FR_moduleNumber );
        RL_module = new Module( ”RL”, Settings.RL_moduleNumber );
        RR_module = new Module( ”RR”, Settings.RR_moduleNumber );

        // TRANSLATION OBJECT
        FL_Trans2d = new Translation2d( Settings.FLx, Settings.FLy );
        FR_Trans2d = new Translation2d( Settings.FRx, Settings.FRy );
        RL_Trans2d = new Translation2d( Settings.RLx, Settings.RLy );
        RR_Trans2d = new Translation2d( Settings.RRx, Settings.RRy );

        // KINEMATICS OBJECT
        Kinematics = new SwerveDriveKinematics( FL_Trans2d, FR_Trans2d, RL_Trans2d, RR_Trans2d );
    }

    public static void Display () {
        SmartDashboard.putNumber(”Robot−vx”, RobotSpeed.vxMetersPerSecond      );
        SmartDashboard.putNumber(”Robot−vy”, RobotSpeed.vyMetersPerSecond      );
        SmartDashboard.putNumber(”Robot−vt”, RobotSpeed.omegaRadiansPerSecond  );

        FL_module.Display();
        FR_module.Display();
        RL_module.Display();
        RR_module.Display();
    }

    public static void UpdateFieldRelative ( double vx, double vy, double vt ) {
        Rotation2d     Rot2d  = Rotation2d.fromDegrees( Navigation.GetYaw() );
        ChassisSpeeds Speeds = ChassisSpeeds.fromFieldRelativeSpeeds( vx, vy, vt, Rot2d );
        Update( Speeds );
    }

    public static void UpdateRobotRelative ( double vx, double vy, double vt ) {
        ChassisSpeeds Speeds = new ChassisSpeeds( vx, vy, vt );
        Update( Speeds );
    }

    private static void Update ( ChassisSpeeds Speeds ) {

        // CALCULATE INDIVIDUAL MODULE STATES
        SwerveModuleState[] ModuleStates = Kinematics.toSwerveModuleStates( Speeds );

        // NORMALIZE WHEEL RATIOS IF ANY SPEED IS ABOVE SPECIFIED MAXIMUM
        SwerveDriveKinematics.desaturateWheelSpeeds( ModuleStates, Settings.MAX_DRIVE_RATIO );

        // UPDATE ROBOT SPEEDS
        RobotSpeed = Kinematics.toChassisSpeeds( ModuleStates );

        // UPDATE EACH MODULE
        FL_module.Update( ModuleStates[0] );
        FR_module.Update( ModuleStates[1] );
        RL_module.Update( ModuleStates[2] );
        RR_module.Update( ModuleStates[3] );
    }

    public static void SetWheelsToHeading ( double Heading ) {
        FL_module.SetToHeading( Heading );
        FR_module.SetToHeading( Heading );
        RL_module.SetToHeading( Heading );
        RR_module.SetToHeading( Heading );
    }

}
```

Track.java

```
package frc.robot.Hardware;

public class Track {

    public static void Track_00 () {
        switch ( Stage.Number ) {
            case 0:
                Autopilot.SetWheelsToHeading( 90 );
                Stage.WaitForDuration( 1.00 );
                break;

            case 1:
                Autopilot.DriveWest( 0.08 );
                Stage.WaitForDuration( 2.00 );
                break;

            default:
                Stage.Last();
                break;
        }
    }

    public static void Track_01 () {
        switch ( Stage.Number ) {
            case 0:
                break;

            default:
                Stage.Last();
                break;
        }
    }

    public static void Track_02 () {
        switch ( Stage.Number ) {
            case 0:
                break;

            default:
                Stage.Last();
                break;
        }
    }

    public static void Track_03 () {
        switch ( Stage.Number ) {
            case 0:
                break;

            default:
                Stage.Last();
                break;
        }
    }

}
```

Default.java (Driver)

```java
package frc.robot.Driver;

import edu.wpi.first.wpilibj.Joystick;
import frc.robot.Hardware.Navigation;
import frc.robot.Hardware.Swerve;
import frc.robot.Mode.Teleop;

public class Default {

    public static void Periodic () {

        // GET VALUES
        Joystick DriveStick = Teleop.DriveStick;
        double   Xratio     = Teleop.Xratio;
        double   Yratio     = Teleop.Yratio;
        double   Tratio     = Teleop.Tratio;

        // SIMPLE JOYSTICK DEADBAND
        if ( Math.abs( Xratio ) < 0.15 ) { Xratio = 0; }
        if ( Math.abs( Yratio ) < 0.15 ) { Yratio = 0; }
        if ( Math.abs( Tratio ) < 0.20 ) { Tratio = 0; }

        if ( DriveStick.getRawButton( 7 ) ) {
            Navigation.Reset();
        }

        // SEND SPEEDS TO SWERVE CLASS
        Swerve.UpdateRobotRelative( Xratio, Yratio, Tratio );
    }

}
```