

Java conventions for objects, methods and variables

Creating objects that are connected to the roboRIO in Java

```
Gyro headingGyro = new AnalogGyro(1);  
double heading = headingGyro.getAngle();
```

Generally all the objects in WPILib that connect to one of the roboRIO breakout boards have one argument in the constructor when created where you specify the channel or port number it is connected to. The above example illustrate the conventions used in WPILib for Java.

1. Creates an AnalogGyro object connected to analog channel 1 and stores its address in "headingGyro".
2. Gets the current heading from the AnalogGyro in degrees and stores it in the variable "heading".

Creating operator interface objects in Java

```
101 Joystick stick = new Joystick(1);  
102  
103 double speed = stick.getX();
```

Generally objects connected to the Driver station PC via USB take a single argument indicating the USB port they are connected to. A single Joystick class is provided which should provide the functionality needed to interface with any joystick or gamepad which works with the FRC Driver Station.

1. Creates a Joystick object connected to USB port 1 on the DS (listed first in the Setup tab of the DS).
2. Gets the current X axis value of the joystick and stores it in the variable "speed".

Java conventions for objects, methods and variables

Class, method and variable naming

| Type of name | Naming rules | Examples |
|-----------------|---|--|
| Class name | Initial upper case letter then camel case (mixed upper/lower case) except acronyms which are all upper case | Victor, SimpleRobot, PWM |
| Method name | Initial lower case letter then camel case | isAutonomous, getAngle |
| Member variable | "m_" followed by the member variable name starting with a lower case letter then camel case | m_deleteSpeedControllers, m_sensitivity |
| Local variable | Initial lower case | targetAngle |

MXP IO Numbering

| Pinout Designations | | | | | |
|---------------------|------------------|----|----|------------------|-----------------|
| C++/Java | | | | | C++/Java |
| DIO 25 | DIO 15 / I2C SDA | 34 | 33 | +3.3V | |
| DIO 24 | DIO 14 / I2C SCL | 32 | 31 | DIO 10 / PWM6 | DIO20/ PWM16 |
| | DGND | 30 | 29 | DIO 9 / PWM5 | DIO19/ PWM15 |
| | DGND | 28 | 27 | DIO 8 / PWM4 | DIO18/ PWM14 |
| DIO23/ PWM19 | DIO 13 / PWM9 | 26 | 25 | DIO 7 / SPI MOSI | DIO17 |
| | DGND | 24 | 23 | DIO 6 / SPI MISO | DIO16 |
| DIO22/ PWM18 | DIO 12 / PWM8 | 22 | 21 | DIO 5 / SPI CLK | DIO15 |
| | DGND | 20 | 19 | DIO 4 / SPI CS | DIO14 |
| DIO21/ PWM17 | DIO 11 / PWM7 | 18 | 17 | DIO 3 / PWM3 | DIO13/ PWM13 |
| | DGND | 16 | 15 | DIO 2 / PWM2 | DIO12/ PWM12 |
| | UART.TX | 14 | 13 | DIO 1 / PWM1 | DIO11/ PWM11 |
| | DGND | 12 | 11 | DIO 0 / PWM0 | DIO10/ PWM10 |
| | UART.RX | 10 | 9 | AI3 | AI7 |
| | DGND | 8 | 7 | AI2 | AI6 |
| | AGND | 6 | 5 | AI1 | AI5 |
| AO1 | AO1 | 4 | 3 | AI0 | AI4 |
| AO0 | AO0 | 2 | 1 | +5V | |

In C++ and Java the numbering for the MXP IO is a continuation of the numbering from the headers, meaning MXP DIO 0 is DIO 10, MXP DIO 1 is DIO 11 and so on. This applies to DIO, PWM

Java conventions for objects, methods and variables

and Analog Input on the MXP. The I2C and SPI buses have enumerations used to indicate which port you are using.