2020 New Command Class

Notes Based on: LINK

Location

- New namespace: "frc2"

Command and subsystem Interfaces

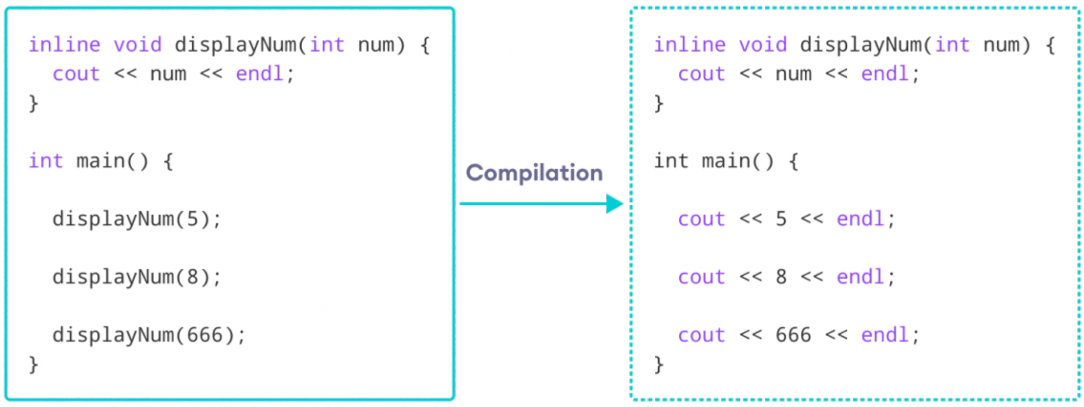
- Command and subsystem are abstract classes. "CommandBase" and "SubsystemBase" abstract base are still provided

Multiple Command Group Classes

- No Command group class but multiple command group classes
    - SequentialCommandGroup
    - ParallelCommandGroup
    - ParallelRaceGroup
    - ParallelDeadlineGroup
    - Details

Inline Command definition

- What is inline function: Inline function basically replaces the code from the function when it is called:



```
inline void displayNum(int num) {
  cout << num << endl;
}

int main() {

  displayNum(5);

  displayNum(8);

  displayNum(666);
}
```

```
inline void displayNum(int num) {
  cout << num << endl;
}

int main() {

  cout << 5 << endl;

  cout << 8 << endl;

  cout << 666 << endl;
}
```

- No longer have to write a command for every case, there's many new inline command designs
  - Specify what constructor will be run by passing subroutine and parameters
  - Could use C++ Lambda expression
  - More Examples on inline commands
    - LINK

Injection of Command Dependencies

- new command class utilizes injection of subsystem dependencies into command so subsystems are not declared globally
  - More Clear
  - More maintainable
  - More reusable
- Specific Structuring: LINK

Command ownership

- Previous framework requires the use of raw pointers which may cause memory leak
- New framework offer ownership management
- Scheduler typically own:
  - Default command

    - Button commands
- Should generally not heap allocate command with "new"

Change to the Scheduler

- "Scheduler" renamed to "CommandScheduler"
- Interruptibility of commands is now the responsibility of the scheduler, not the commands, and can be specified during the call to "schedule"
- Can pass action to scheduler
    - Useful in even logging

Change in subsystem

- Subsystem is an interface
- Closest equivalent to old "Subsystem" is new "SubsystemBase"
- removed "InitDefaultCommand". Default registered directly from "CommandScheduler"
- New "SetDefaultCommand" is wrapped in "CommandScheduler"
- Subsystem no long "Know about" the command requiring them instead it is handled exclusively by "CommandScheduler"

Change in Command

- Command is an interface
- Closest equivalent to old "Command" is new "CommandBase"
- Command no longer handles scheduling
- "Interrupted()" is rolled into "end()"
- "requires()" renamed to "addRequirements()"
- "void setRunsWhenDisabled(boolean disabled)" replaced by an overrideable "runsWhenDisabled()"
- "void setInterruptible(boolean interruptible)" removed
- Several new methods added:
    - withTimeout
    - withInterrupt

    - And more
- In order to allow the decorator to work with command ownership model, a CRTP model is used via "CommandHelper" class.
- Any user-defined Command subclass "Foo" must extend "CommandHelper<Foo, Base>" where Base is the desired base class.

Changes to PID System / PID Command

- "PIDController" injected through constructor. Could be modifier after construction with "getController"
- "PIDController" Largely for inline uses
- If wish to use "PIDCommand" more “traditionally,” overriding the protected "returnPIDInput()" and "usePIDOutput(double output)" methods, replaced by modifying the protected "m_measurement" and "m_useOutput" fields
- rather than calling "setSetpoint", can modify the protected "m_setpoint" field