Basic OpModes Complete Code Walkthrough

This document provides a comprehensive, line-by-line explanation of the three fundamental FTC OpModes. We will analyze the Java concepts, logic, and execution flow for each program.

---

1. `BasicOpMode_Linear.java`

The Standard Procedural OpMode

This program runs sequentially (line-by-line) inside a single method. It is the easiest to understand for beginners.

1.1 Class Definition & Members

```
@TeleOp(name="Basic: Linear OpMode", group="Linear OpMode")

public class BasicOpMode_Linear extends LinearOpMode {

    private ElapsedTime runtime = new ElapsedTime();

    private DcMotor leftDrive = null;

    private DcMotor rightDrive = null;
```

- `@TeleOp` : This **Annotation** registers the program with the Robot Controller. It will appear in the "TeleOp" list on the phone.
- `extends LinearOpMode` : **Inheritance**. Our class inherits all the features of `LinearOpMode` (like `hardwareMap` and `telemetry`).
- `private DcMotor ...` : **Encapsulation**. We declare our hardware objects as `private` fields so they can be used anywhere in this class, but not outside it.

1.2 The `runOpMode()` Method

```
@Override

public void runOpMode() {

    telemetry.addData("Status", "Initialized");
```

```
    telemetry.update();
```

- `@Override` : **Polymorphism**. We are replacing the empty `runOpMode()` from the parent class with our own instructions.
- `telemetry.update()` : In a LinearOpMode, you MUST call this to send data to the driver station.

1.3 Hardware Initialization

```
    leftDrive  = hardwareMap.get(DcMotor.class, "left_drive");
    rightDrive = hardwareMap.get(DcMotor.class, "right_drive");

    leftDrive.setDirection(DcMotor.Direction.REVERSE);
    rightDrive.setDirection(DcMotor.Direction.FORWARD);
```

- `hardwareMap.get(...)` : This retrieves the configured motor from the phone's configuration file.
- `setDirection(...)` : We reverse one motor because the motors are mounted on opposite sides of the robot (mirror image). If we didn't do this, the robot would spin when we tried to drive forward.

1.4 Waiting for Start

```
    waitForStart();
    runtime.reset();
```

- `waitForStart()` : A **Blocking Call**. The program pauses here and waits for the driver to press "Start".
- `runtime.reset()` : Resets the game timer to 0.

1.5 The Main Loop

```
    while (opModeIsActive()) {
```

```
        double drive = -gamepad1.left_stick_y;

        double turn  =  gamepad1.right_stick_x;
```

- `while (opModeIsActive())`: The **Control Loop**. This runs hundreds of times per second until the match ends.
- `double`: We use floating-point numbers because joystick values are continuous (-1.0 to 1.0).
- **Negative Sign (-)**: The Y-axis on gamepads is reversed (Up is -1, Down is +1). We negate it so Up becomes Positive.

1.6 Calculating Power (POV Mode)

```
        double leftPower    = Range.clip(drive + turn, -1.0, 1.0);

        double rightPower   = Range.clip(drive - turn, -1.0, 1.0);
```

- **The Algorithm**:

* To turn right (`turn > 0`), we add power to the left wheel and subtract from the right.

- `Range.clip(...)`: **Safety**. If `drive` is 1.0 and `turn` is 1.0, the result is 2.0. This function forces the number to stay between -1.0 and 1.0.

1.7 Sending Commands

```
        leftDrive.setPower(leftPower);

        rightDrive.setPower(rightPower);


        telemetry.addData("Status", "Run Time: " + runtime.toString());

        telemetry.update();

    }

}
```

- `setPower(...)` : The physical action. The motor controller receives the command.
- `telemetry.update()` : Sends the loop data to the phone screen.

---

2. `BasicOpMode_Iterative.java`

The Event-Driven OpMode

This program does NOT have a `while` loop. Instead, the system calls specific methods repeatedly.

2.1 The Lifecycle Methods

```
@Override
public void init() {
    // Runs ONCE when you hit INIT
}

@Override
public void init_loop() {
    // Runs REPEATEDLY during INIT (e.g., for vision targeting)
}

@Override
public void start() {
    // Runs ONCE when you hit START
    runtime.reset();
}
```

- **Event-Driven**: The robot controller acts as the "Main Loop". It calls your methods at the right time.

2.2 The `loop()` Method

```
@Override
public void loop() {
    double drive = -gamepad1.left_stick_y;
    double turn  =  gamepad1.right_stick_x;
    // ... math ...

    leftDrive.setPower(leftPower);
    rightDrive.setPower(rightPower);
}
```

- **Non-Blocking**: You cannot use `Thread.sleep()` or `while()` here. This method must finish instantly so the system can call it again.
- **Implicit Telemetry**: Notice there is **NO** `telemetry.update()`. The system handles it automatically after `loop()` finishes.

---

3. `BasicOmniOpMode_Linear.java`

The Holonomic (Mecanum) OpMode

This program introduces vector math for robots that can move sideways.

3.1 Three Axes of Control

```
double axial   = -gamepad1.left_stick_y;  // Forward/Backward
double lateral =  gamepad1.left_stick_x;  // Strafe Left/Right
double yaw     =  gamepad1.right_stick_x; // Rotate
```

- **Abstraction**: We separate the *intent* (Move Forward) from the *mechanism* (Spin wheels).

3.2 The Mixing Formula

```
double frontLeftPower  = axial + lateral + yaw;
double frontRightPower = axial - lateral - yaw;
double backLeftPower   = axial - lateral + yaw;
double backRightPower  = axial + lateral - yaw;
```

- **Vector Addition**:

* **Forward**: All wheels +

* **Strafe Right**: FL & BR are +, FR & BL are - (The wheels fight each other to create sideways force).

* **Turn Right**: Left wheels +, Right wheels -

3.3 Normalization (Advanced Math)

```
max = Math.max(Math.abs(frontLeftPower), Math.abs(frontRightPower));
max = Math.max(max, Math.abs(backLeftPower));
max = Math.max(max, Math.abs(backRightPower));

if (max > 1.0) {
    frontLeftPower  /= max;
    frontRightPower /= max;
    backLeftPower   /= max;
    backRightPower  /= max;
}
```

- **The Problem**: If you go full speed Forward (1.0) and Turn (1.0), one motor wants 2.0 power.

- **Why not Clip?** Clipping changes the ratio. If Left=2.0 and Right=0.5, clipping makes it 1.0 and 0.5 (Ratio 2:1). The original ratio was 4:1. The robot would turn at the wrong angle.
- **The Solution**: Divide everything by the max (2.0).

* Left becomes 1.0.

* Right becomes 0.25.

* Ratio is 4:1. **Direction is preserved.**

Summary of Java Concepts

| Concept | Explanation | Example |

| :--- | :--- | :--- |

| **Inheritance** | Using code from a parent class. | `extends LinearOpMode` |

| **Polymorphism** | Customizing a parent's method. | `@Override public void runOpMode()` |

| **Encapsulation** | Hiding data inside a class. | `private DcMotor leftDrive` |

| **Blocking Call** | Pausing execution. | `waitForStart()` |

| **Control Loop** | Repeating code continuously. | `while (opModeIsActive())` |

| **Telemetry** | Sending data to the phone. | `telemetry.addData(...)` |

| **Hardware Mapping** | Linking to physical hardware. | `hardwareMap.get(...)` |

| **Motor Direction** | Controlling motor rotation. | `setDirection(...)` |

| **Safety** | Preventing unsafe values. | `Range.clip(...)` |

| **Event-Driven** | Responding to system events. | `init()`, `init_loop()`, `start()`, `loop()` |

| **Vector Math** | Calculating complex movements. | `axial`, `lateral`, `yaw` |

| **Normalization** | Adjusting ratios to prevent overdrive. | `Math.max(...)` |

| **Runtime** | Tracking elapsed time. | `runtime.reset()` |

| **Gamepad Input** | Controlling with gamepad. | `gamepad1.left_stick_y` |

| **Motor Power** | Controlling motor speed. | `setPower(...)` |