FTC – JAVA PROGRAMMING

Workshop 2015

Eric Weber
FRC: 1322, FTC: 5954 & 7032

Java History

- First appeared in 1995
- Sun Microsystems creates and maintains the core language
- Community involvement is very high in the development
- Appears in many small devices
- Want college credit in Computer Science?
  - Java is the standard language for AP CS courses
- Most importantly, it is currently the gateway into other languages
  - Know java? You know C, C++, C#, Python, Ruby, Pascal and many others with minimal understanding

What you will need?

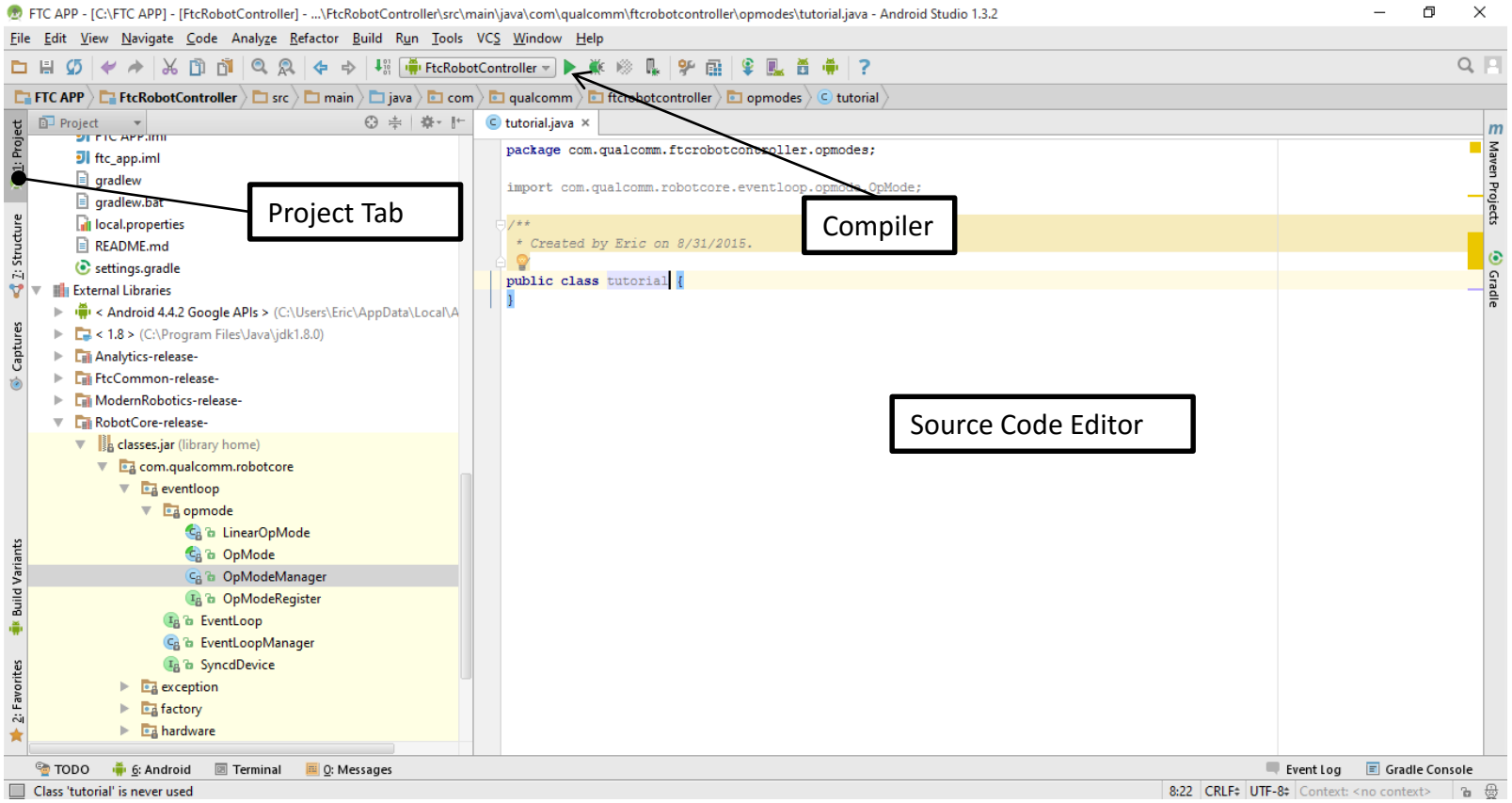
- Android Studio
  - http://developer.android.com/sdk/index.html#top
- FTC_App
  - https://github.com/ftctechnh/ftc_app/archive/master.zip
- Tutorial for setting up phones
  - https://github.com/ftctechnh/ftc_app/blob/master/doc/tutorial/FTCTraining_Manual.pdf

Kettering FTC Workshop

If you haven't followed the instructions on http://paws.kettering.edu/~webe3546/

Start Downloading
The process will take a long time to complete

Introducing the UI



Important Definitions

- **IDE (Integrated Development Environment):**
  - Android Studio itself is an IDE. It contains a source code editor, compiler, and a debugger all in one.
- **OP Modes:** define how our robots behave
  - Teleop and Autonomous modes are now called OP Modes
- **Keywords:** Reserved words that Java requires, and cannot be used as an unique name

FTC
FIRST® Tech Challenge

TELEOP EXAMPLE CODE

Will work with the robot being built during this workshop.

FIRST® Tech Challenge

Code Objective:

Tutorial.java

```
package com.qualcomm.ftcrobotcontroller.opmodes;

import com.qualcomm.robotcore.eventloop.opmode.OpMode;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorController;

/**
 * Created by webe3546 on 9/3/2015.
 */
public class Tutorial extends OpMode {

    /*****************************************
     *  Motor Controllers
     */
    private DcMotorController dc_drive_controller;

    /*****************************************
     * Motors Used
     */
    private DcMotor dc_drive_left;
    private DcMotor dc_drive_right;

    @Override
    public void init() {
        dc_drive_controller = hardwareMap.dcMotorController.get("drive_controller");
        dc_drive_left = hardwareMap.dcMotor.get("drive_left");
        dc_drive_right = hardwareMap.dcMotor.get("drive_right");
    }

    @Override
    public void loop() {
        dc_drive_left.setPower(gamepad1.left_stick_y);
        dc_drive_right.setPower(gamepad1.right_stick_y);
    }
}
```

Don’t write this down yet. We will cover this line by line.

Teleop Mode Example:

- This code is strictly for a simple tele OP mode
- Not optimal for programming a robot with an autonomous mode
- Equivalent to a ‘Hello World’ for the robot that is being built in the class rooms upstairs

Creating an OP Mode:

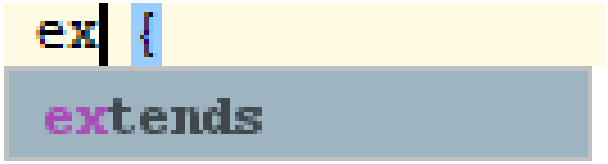
1) In the Project Tree Navigate:
   - FTC APP -> FtcRobotController -> src -> main -> com.qualcomm.ftcrobotcontroller -> opmodes
3) *Right Click* on the Folder
4) Go to *"New"* -> *"Java Class"*
5) Give it a name
6) Click *"OK"*

Edit Class Definition:

- Once we have created our OP Mode, we need to edit a line immediately.
- Please add “extends OpMode” between the *name of class* and the “{“

```
public class Tutorial extends OpMode {
```

Anyone Notice This?



- Then you may auto complete with the selected word below
- Press the *"Tab"* key to allow completion.
- A benefit of using an *IDE* allows easier functionality

Important Notes:

- First we are defining a *public class*
  - Classes defines data formatting and procedures
  - *Public* defines how it may be accessed
    - In this case, anywhere
- Second we have a unique name for these *classes*
  - Must be unique and not be a *keyword*
- Third we are extending a *parent class* (*Inheritance*)
  - We are directly adding onto a class already made
  - We also gain the functionality of this class

Define Properties

- Next we will enter in the following below
- These are what are called *properties* or *fields*
  - From here on out, we will refer to them as properties

```
public class Tutorial extends OpMode {

    /*****************************************
     *  Motor Controllers
     */
    private DcMotorController dc_drive_controller;

    /*****************************************
     * Motors Used
     */
    private DcMotor dc_drive_left;
    private DcMotor dc_drive_right;
```

Important Notes:

- Properties allow us to define data to be used
- In this case we are defining:
  - 1 DcMotorController (a class soon to be an object)
  - 2 DcMotor (another class soon to be an object)
- Later we will be able to effect the values of the DC Motors
- In non-OOP languages, these are also known as variables (RobotC)
- If you want more, you have to define more

Our first Method

- Methods allow us to perform tasks
- Please enter the next lines after our properties:

```
@Override
public void init() {
    dc_drive_controller = hardwareMap.dcMotorController.get("drive_controller");
    dc_drive_left = hardwareMap.dcMotor.get("drive_left");
    dc_drive_right = hardwareMap.dcMotor.get("drive_right");
}
```

FTC
FIRST® Tech Challenge

Important Notes:

- First off, *@Override* allows us to over write a previous method from OpMode.
  - This is one of two methods that MUST be overridden.
  - This will always be the first method called once the ARM button is pressed on the robot controller.
- Second, we are assigning actual objects to the properties we already have defined

Important Notes:

- Third, what is hardwareMap?
  - It is an object that contains all the hardware mapping as defined by the configuration files on your Robot Controller app
  - Everything stated by your robot configuration file will be here. This makes setting up your configuration correctly and translate it **EXACTLY** into your java code.

Our Second Method

- The second *method* we must override is the loop() *method*.
- Please enter the next lines after our previous *method*.

```
@Override
public void loop() {
    dc_drive_left.setPower(gamepad1.left_stick_y);
    dc_drive_right.setPower(gamepad1.right_stick_y);
}
```

Important Notes:

- This *method* is called every time the robot cycles (approx. 20ms give or take)
- Not where to apply a loop
- Since a part of OpMode, this will be consistent with autonomous OpModes as well

Final step: Register your OpMode

- We need to finalize the app by registering our OpMode with the rest of the program.
- Navigate through the project tab to: ftc_app-master -> FtcRobotController -> src -> main -> java -> com -> qualcomm -> ftcrobotcontroller -> opmodes -> FtcOpModeRegister
- Under the register method, type: manager.register("Tutorial", Tutorial.class);

```
public void register(OpModeManager manager) {
    /*
    * The NullOp op mode
    */
    manager.register("NullOp", NullOp.class);

    /*
    *  Our Op modes
    */
    manager.register("Tutorial", Tutorial.class );
```

Important Notes:

- To be able to select your OpMode, it needs to be added to a list.
- I have already trimmed down the OpModes that were used as tutorials.
- NullOp will do nothing.
  - Good to keep due to any issues that arise.

FTC
FIRST® Tech Challenge

Important Definitions:

- **Class:** Defines data format and procedures
- **Properties:** Variables defined by the class
- **Methods:** Procedures that work on inputs or properties
- **Inheritance:** The ability to extend a class to include more functionality (*methods*) or data (*properties*)
- **Overriding:** The ability to take a method and change it to give different functionality
- **Constructor:** As an object is created, a special method is always called immediately.

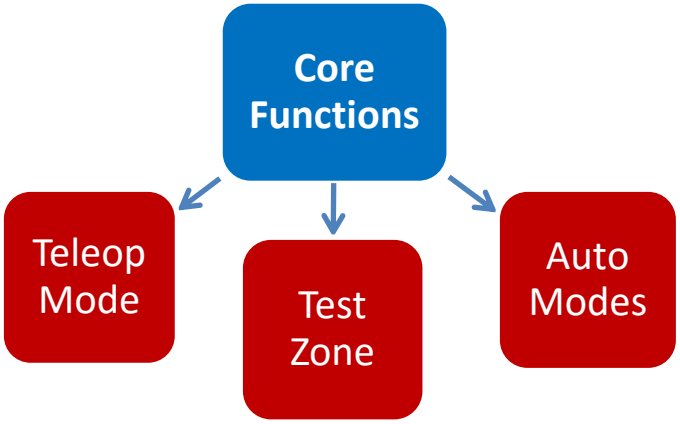
FTC
FIRST® Tech Challenge

STRUCTURAL SUGGESTIONS

Ideas to extend your code from Teleop to Autonomous modes

Hierarchy

- Object Oriented Programming's Greatest asset is reusability and extensibility.
- Better to define a robot by its actions, then control it through those actions.



Key Ideas on Inheritance

- Inheritance Properties:
  - Extending a base class forces the base class to exist, giving us those methods as well
    - Think drive systems, arms, sensors, or timers required
    - But know about Private, Public, and Protected
  - We only need to override the operation modes we want.
  - For instance, leave initialization for the base class, only work with loop() in the actual OpModes.

Important Keys to Note:

- Methods with inheritance
- Overriding (Virtual Methods)
- OpModes
- Encapsulation

NOW AN OPEN DISCUSSION.

Questions and Suggestions?

What would you like to see

FTC
FIRST® Tech Challenge

AUTONOMOUS MODE

A method to accomplish tasks in Autonomous mode

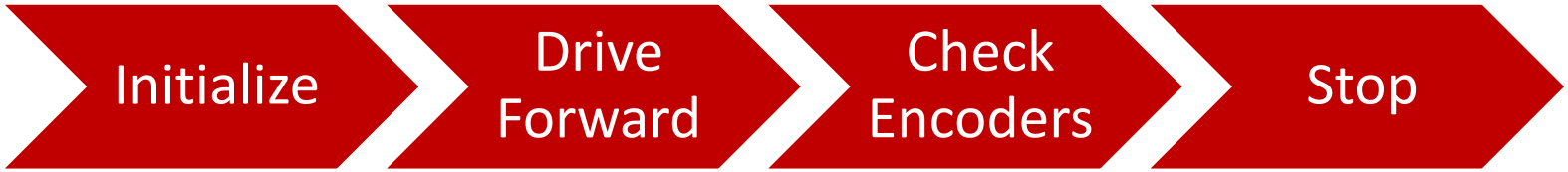
FTC
FIRST® Tech Challenge

State Machines

- For this style of programming, State Machines are the suggested method.
- Review of State Machines:
  - Idea of states: Set of instructions unique to a phase of a program
  - States define what the robot is to do
  - Redefine outputs
  - Read inputs to trigger next state

State Machines

- Requirements of a state machine:
  - A state variable (usually an enumeration)
  - A state selector (always a case-switch operator)
  - State triggers (sensors or timers)
  - An initial state



Enumerations:

- Enumerations are unique names with values defined behind them
- Common examples include compass directions (values of NORTH, SOUTH, EAST, and WEST)
- Place above the actual OpMode

```
enum State {
    INITIALIZE, MOVE, CHECK, STOP
}
```

Switch and Case Structure:

- Allows for multiple cases or states to make different operations
- Selector can take Enumeration's, Integer's commonly

Switch Structure – Refers to all cases

Selector – Selects whatever value is entered

Case – Different states, as selected by the selector

Break- Escapes out of structure

Default- If no valid case exists, default will always be used

```
switch (state_s){
    case INITIALIZE:
        state_s = State.MOVE;
        resetTime();
        break;
    case MOVE:
        this.DriveRobotTank(1.0f, 1.0f);
        state_s = State.CHECK;
        break;
    case CHECK:
        if (this.getTime() > 1000) state_s = State.STOP;
        break;
    case STOP:
        this.StopDrive();
        break;
    default:
        state_s = State.STOP;
        break;
}
```

Triggers:

- Usually done by an if statement
  - If statements are like case's, but can easier to define with logical statements
- Causes a change in our state variable

```
if (this.getTime() > 1000) state_s = State.STOP;
```

Putting it Together:

- We get the following OpMode:

```
package com.qualcomm.ftcrobotcontroller.opmodes;

/**...*/

enum State {
    INITIALIZE, MOVE, CHECK, STOP
}

public class TutorialAuton extends TutorialBot {
    State state_s;

    @Override
    public void start() {
        state_s = State.INITIALIZE;
    }
```

```
@Override
public void loop() {
    switch (state_s){
        case INITIALIZE:
            state_s = State.MOVE;
            resetTime();
            break;
        case MOVE:
            this.DriveRobotTank(1.0f, 1.0f);
            state_s = State.CHECK;
            break;
        case CHECK:
            if (this.getTime() > 1000) state_s = State.STOP;
            break;
        case STOP:
            this.StopDrive();
            break;
        default:
            state_s = State.STOP;
            break;
    }
}
```

FTC
FIRST® Tech Challenge

FINAL REMARKS & QUESTIONS

Thank You

Now get out their and program!