Introduction à la programmation Java en Compétition de robotique *FIRST*

Structure Command based

Régis Bekale
Robotique *FIRST* Québec,
Automne 2019

ROBOTIQUE
FIRST
QUÉBEC

Plan de la présentaton (1/4)

- Structure générale d'un robot typique de la Compétition de robotique *FIRST*
- Comment le programmer? (les solutions possibles)
  - *Simple Robot*
  - *Timed Robot*
  - *Skeleton Robot*
  - *Command Robot*
- Pour chaque solution: avantages et inconvénients

Plan de la présentaton (2/4)

- Structure générale d'un projet en mode *command based* (~2009-2019)
  - *RobotMap*
  - Sous-systèmes (*Subsystems*)
  - Commandes (*Commands*)
  - Interface Opérateur (*OI*)
  - Robot

Plan de la présentaton (3/4)

- Structure générale d’un projet en mode *command based* (2020-…)
  - *Constants*
  - *RobotContainer*
  - *Robot*
  - *Subsystems*
  - *Commands*

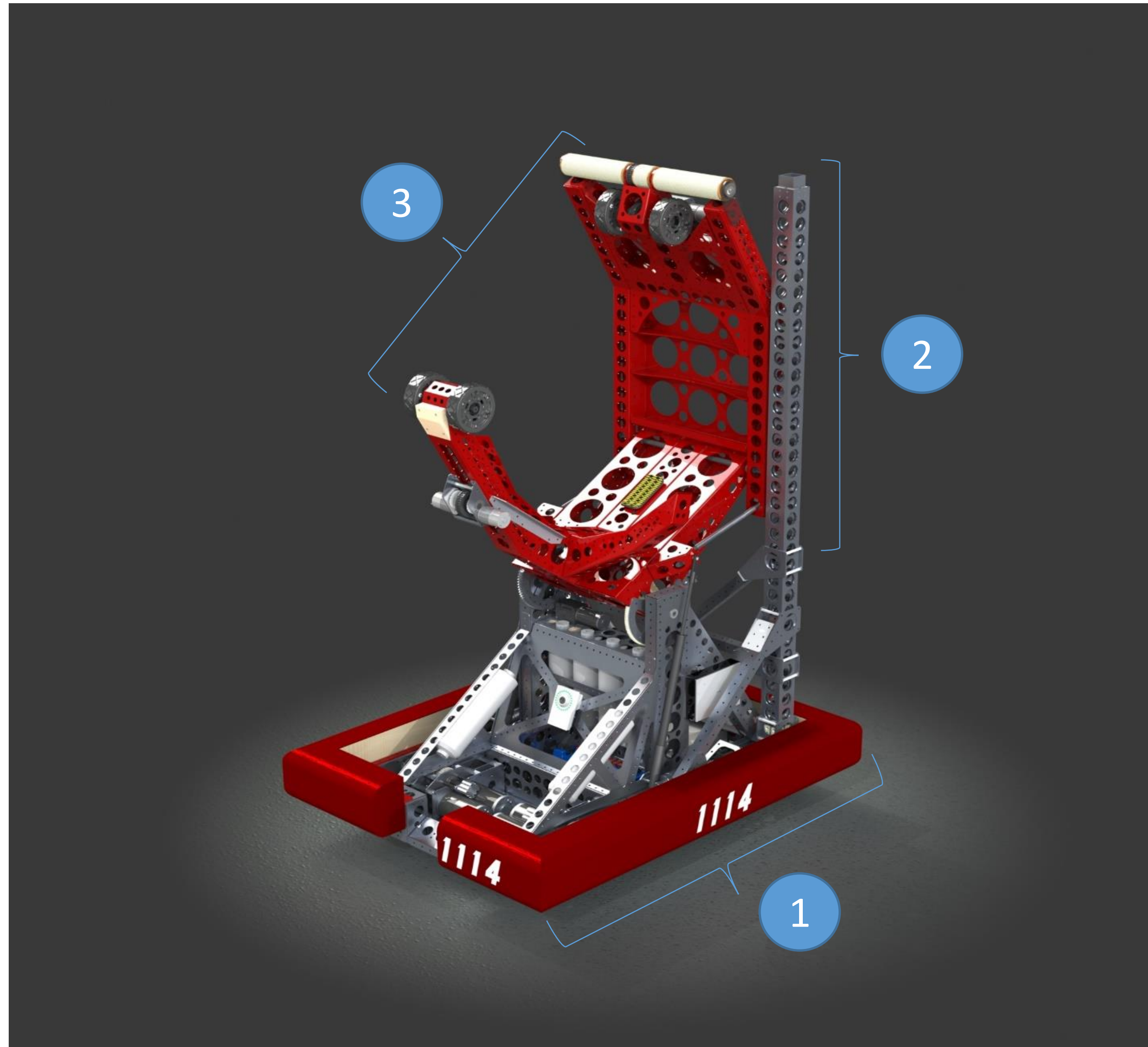
ROBOTIQUE FIRST QUÉBEC

Plan de la présentaton (4/4)

- Librairies disponibles
  - WPILIB
  - CTRE
  - SPARK
  - NAVX
- Conseils pratiques
- Ressources

ROBOTIQUE FIRST QUÉBEC

Structure générale d'un robot de la Compétition de robotique *FIRST*



1 Plateforme pilotable

2 Mécanisme élévateur

3 Mécanisme ramasseur-lanceur

Problème:
Comment programmer ce robot?

ROBOTIQUE FIRST QUÉBEC

Structures disponibles de programmes en WPILIB

- *Iterative Robot*
- *Timed Robot*
- *Timed Skeleton*
- *Command Robot*

Avantages et inconvénients

IterativeRobot

TimedRobot

TimedSkeleton

- ❑ Bon pour un programme simple comme initier les jeunes ou pour tester une simple fonctionnalité.
- ❑ Pas recommendés pour un code de compétition: le programme peut s'allonger rapidement, travail en groupe difficile.

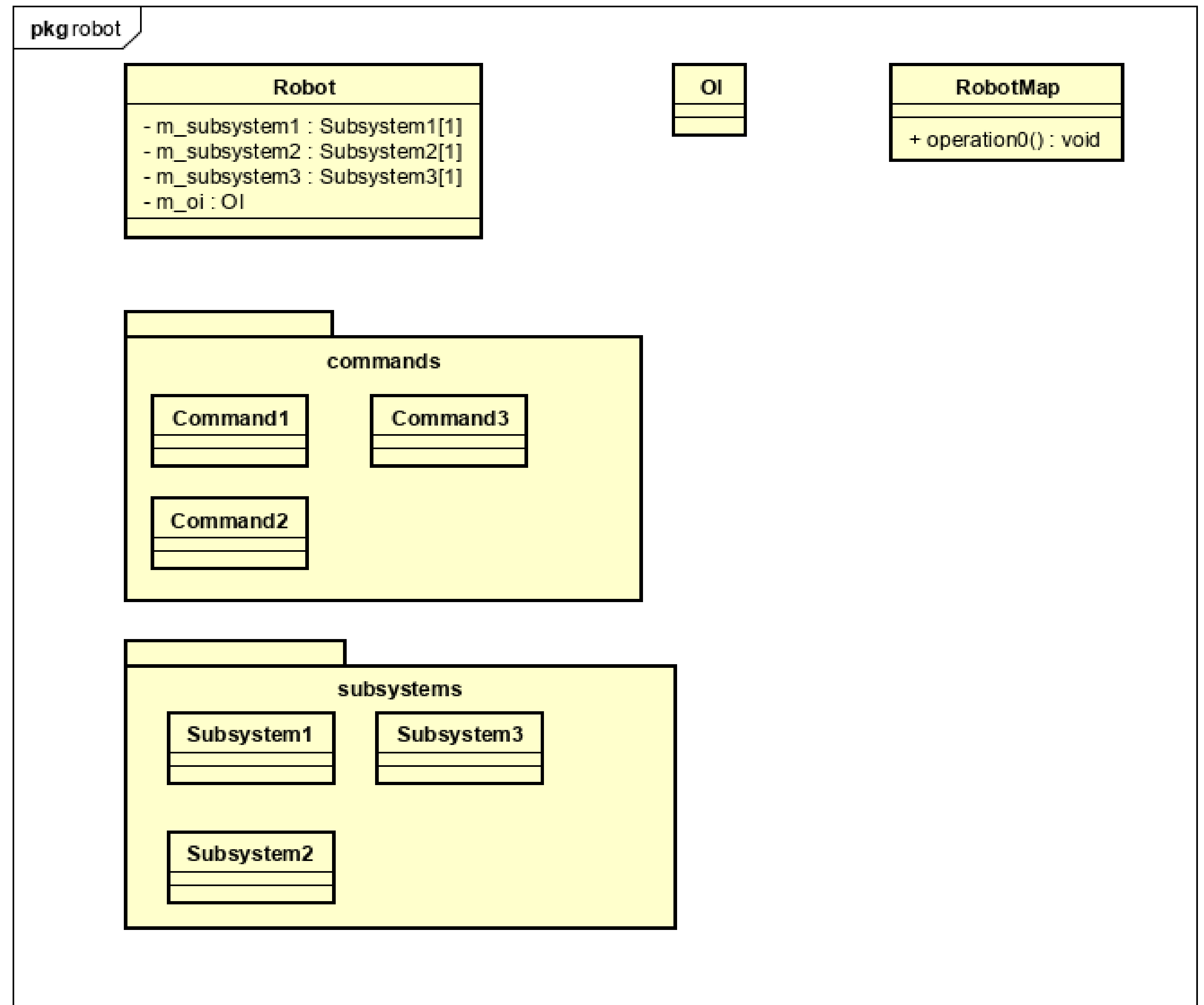
Command Based

- ❑ Bon pour un code complexe avec plusieurs modules. Facilite le travail en groupe.
- ❑ Facile à modifier en compétition (modes autonomes)
- ❑ Pas recommandé pour tester de simples fonctionnalités comme le fonctionnement d'un capteur
- ❑ Pas recommandé dans les premières phases d'initiation des jeunes.

Command Based (2009-2019)

Structure *Command Based*

- ❑Robot
- ❑OI
- ❑RobotMap
- ❑Command
- ❑Subsystem



ROBOTIQUE FIRST QUÉBEC

Robot

Programme principal du projet. Utilise tous les objets correspondant aux ressources:

Drivetrain, shooter, arm, elevator, feeder, vision, joystick, etc.

Rassemble toutes les méthodes nécessaires au déroulement d'un match. Principalement:

Mode Autonome, mode téléopéré, mode test

ROBOTIQUE FIRST QUÉBEC

Robot Class

- public class Robot extends TimedRobot {
- public void robotInit() { }
- public void robotPeriodic() { }
- public void disabledInit() { }
- public void disabledPeriodic() { }
- public void autonomousInit() { }
- public void autonomousPeriodic(){ }
- public void teleopInit() { }
- public void teleopPeriodic(){ }
- public void testPeriodic(){ }
- }

ROBOTIQUE FIRST QUÉBEC

OI (*Operator Interface*)

Est le lieu où l'on déclare les controlleurs (Joystick, gamepad, etc.) nécessaires pour contrôler le robot.

C'est aussi le lieu où sont déclarés les boutons servant à activer les commandes du robot.

Couplage Bouton - Commande

ROBOTIQUE FIRST QUÉBEC

IO Class

- public class OI {

```
public Joystick stick;
public OI() {
  JoystickButton driveButton = new
        JoystickButton(stick, 1);
  driveButton.whenPressed(new
        DriveCommand());
  JoystickButton stopButton = new
        JoystickButton(stick, 2);
  stopButton.whenPressed(new
        StopCommand());
}
```

- }

ROBOTIQUE FIRST QUÉBEC

RobotMap

Contient l'inventaire de l'assignation des ports de tous les composants du robot.

Capteurs, actuateurs.

Permet d'avoir tous les ports rassemblés au même endroit, facilite le déverminage.

ROBOTIQUE FIRST QUÉBEC

RobotMap Class

```
public class RobotMap {
    public static final int leftMotor = 0;
    public static final int rightMotor = 1;
}
```

Commande (Command)

Les commandes contrôlent les sous-systèmes.

Chaque commande s'exécute de façon répétitive jusqu'à la fin de la réalisation de la fonctionnalité attendue sauf si la commande est interrompue par un évènement extérieur.

Des commandes par défaut sont définies pour des sous-systèmes sollicitées en absence de toute autre commande, particulièrement important pour la plateforme pilotable. Définissent le fonctionnement par défaut du sous-système.

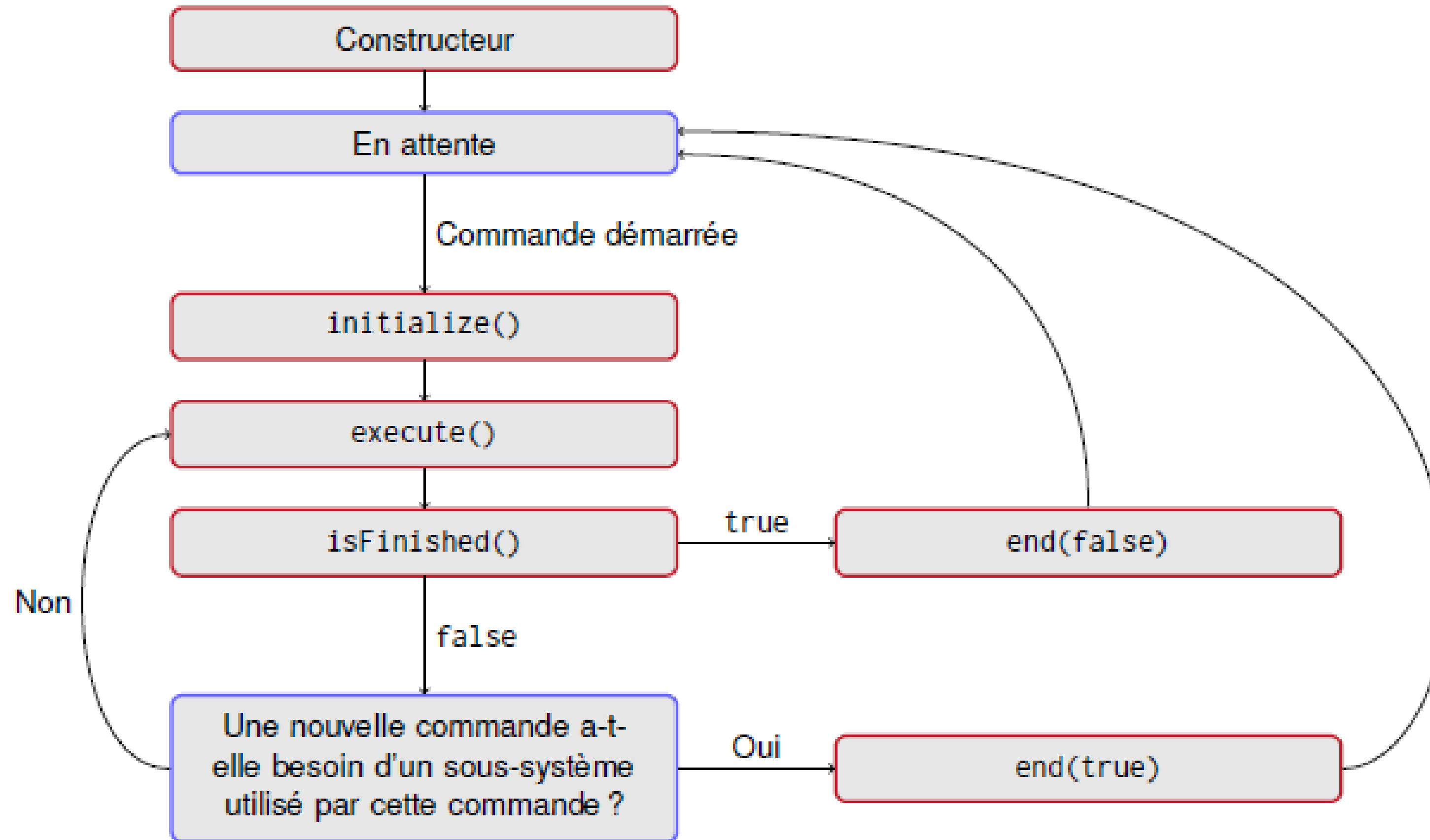
ROBOTIQUE FIRST QUÉBEC

Command Class (Simple)

- public class ExampleCommand extends Command {
-     public ExampleCommand() { }
-     protected void initialize() { }
- protected void execute() { }
- protected boolean isFinished() { }
- protected void end() { }
- protected void interrupted() { }
- }

ROBOTIQUE FIRST QUÉBEC

Cycle de vie d’une commande



Gracieuseté d’Etienne Beaulac, Mentor FRC 5528 Ultime

ROBOTIQUE FIRST QUÉBEC

CommandGroup Class (Complexe ou composée)

- public class ExampleCommandG extends CommandGroup {
  - //addSequential(new Command1());
  - //addSequential(new Command2());

  - //addParallel(new Command1());
  - //addSequential(new Command2());
- }

Séparation des commandes

- Commandes pour le mode téléopéré
- Commandes pour le mode autonome (important de s’assurer des conditions de fin)
- Commandes avec PID , sans PID

ROBOTIQUE FIRST QUÉBEC

Drive Command

- public class DriveCommand extends Command {
- public DriveCommand() {
- requires(Robot.drivetrain);
- }
- public void execute() {
- Robot.drivetrain.drive(0.5, 0);
- }
- }

ROBOTIQUE FIRST QUÉBEC

Sous-Système (Subsystem)

Par sous-système on désigne tout système fonctionnant indépendamment des autres parties du robot, comme :

Drivetrain, shooter, arm, elevator, feeder, vision, etc.

Rassemble tous les objets relatifs aux capteurs et actuateurs nécessaires au fonctionnement du sous-système.

Encodeurs, moteurs, gyroscope, etc.

Contient la définition de toutes les fonctionnalités attendues du sous-système.

Subsystem Class

- public class ExampleSubsystem extends Subsystem {
- public void initDefaultCommand() {
- setDefaultCommand(new ExampleCommand());
- }
- }

Drivetrain Subsystem

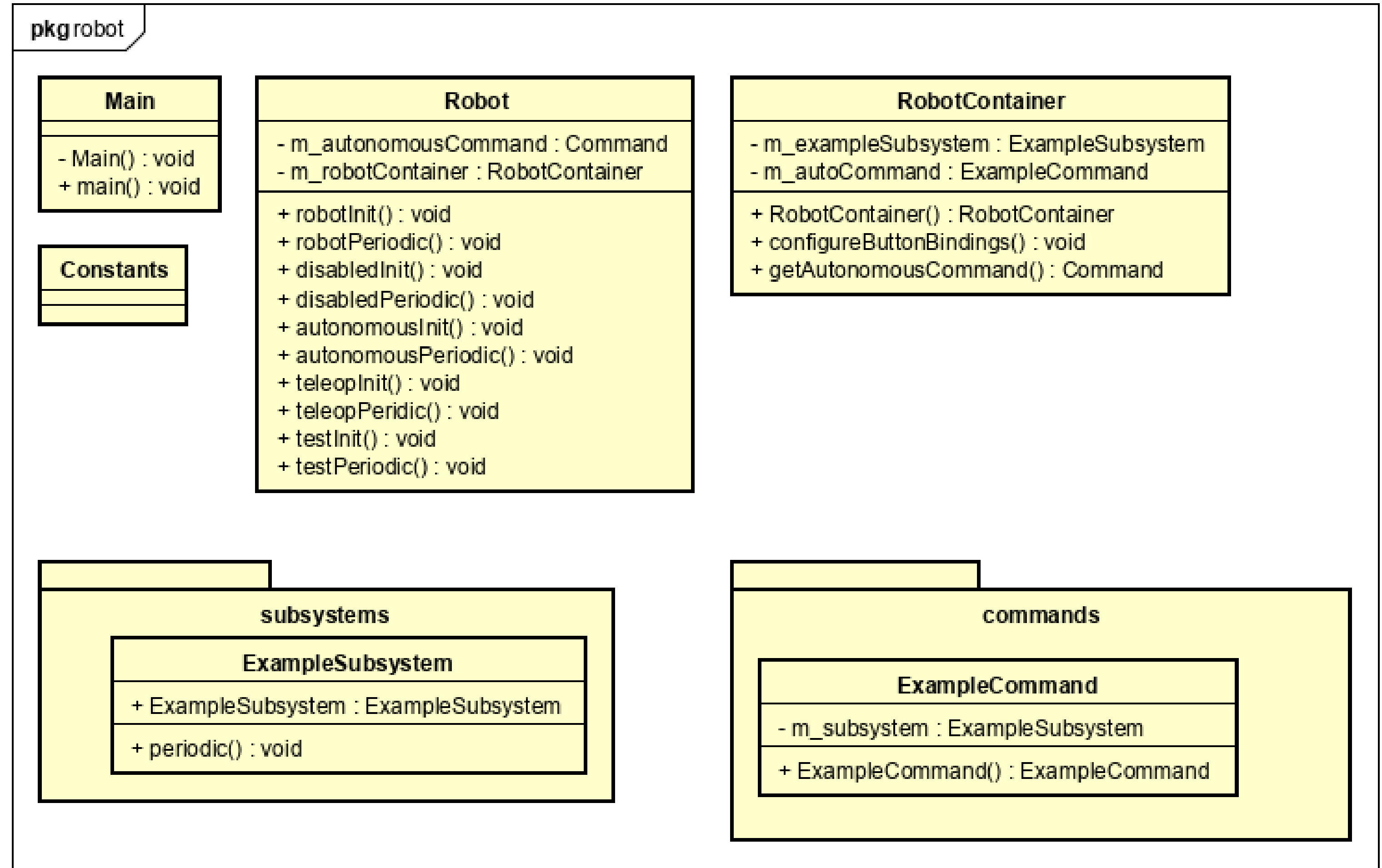
- public class Drivetrain extends Subsystem {
- private Talon left = new Talon(0);
- private Talon right = new Talon(1);
- private DifferentialDrive drive = new
- DifferentialDrive(left, right);
- public void drive(double y, double x) {
- drive.arcadeDrive(y, x);
- }
- }

ROBOTIQUE FIRST QUÉBEC

Command Based (2020-…)

Structure *Command Based* 2020

- ❑Robot
- ❑RobotContainer
- ❑Constants
- ❑Main
- ❑Subsystems
- ❑Commands



ROBOTIQUE FIRST QUÉBEC

Class Constants

```
public final class Constants {
    public static final class DriveConstants {
    public static final int kLeftMotorFront = 0;
    public static final int kLeftMotorRear = 1;
    public static final int kRightMotorFront = 2;
    public static final int kRightMotorRear = 3;
}
public static final class ShooterConstants {
    public static final int kShooterMotor = 4;
}

public static final class AutoConstants {
    public static final int kAutoTimeoutSeconds = 4;
}
```

```
public static final class OIConstants {
    public static final int kDriverControllerPort = 1;
}
}
```

ROBOTIQUE FIRST QUÉBEC

Class RobotContainer

```
public  class RobotContainer {
    private final  Drivetrain  m_drivetrain = new
                                          Drivetrain();
    private final  Shooter m_shooter = new
                                          Shooter() ;
    private final Joystick m_joystick = new
                    Joystick(kDriverControllerPort);

   public RobotContainer(){
     m_drivetrain.setDefaultCommand(new
    ArcadeCommand(() -> m_joystick.getY(), () ->
                    m_joystick.getX(), m_drivetrain));
     configureButtonBindings();
    }

public void configureButtonBindings ( ) {
   final JoystickButton shoot = new
                       JoystickButton(m_joystick, 1);
   shoot.whenPressed(new
                       ShootCommand(m_shooter));
 }
}
```

ROBOTIQUE FIRST QUÉBEC

Class DriveTrain (Subsystem)

```
public  class DriveTrain extends SubsystemBase {
    private final  SpeedControllerGroup  m_left ;
    private final  SpeedControllerGroup  m_right ;
    private final  DifferentialDrive  m_drive ;

    public  DriveTrain {
       m_left = new SpeedControllerGroup(new
       PWMVictorSPX(kLeftMotorFront), new
       PWMVictorSPX(kLeftMotorRear));

        m_right = new SpeedControllerGroup(new
       PWMVictorSPX(kLeftMotorFront), new
       PWMVictorSPX(kLeftMotorRear));
     }

public void arcadeDrive (double forward, double turn)
{
 m_drive.arcadeDrive(forward, turn);
 }
}
```

Class ArcadeCommand (Commande simple)

```
public  class ArcadeCommand extends CommandBase
{
    private final  DriveTrain  m_drivetrain ;
    private final  DoubleSupplier m_forward;
    private final  DoubleSupplier m_turn;
    public  ArcadeCommand (DoubleSupplier fwd,
             DoubleSupplier tur, DriveTrain drivetrain ) {
      m_drivetrain = drivetrain;
      m_forward = fwd;
      m_turn  =   tur;
      addRequirements(m_drivetrain);
       }
      @Override
      public void execute(){
     m_drivetrain.arcadeDrive(m_forward.getAsDouble(),
                m_turn.getAsDouble()); }
@Override
      public void isFinished(){
          return false;
       }
@Override
      public void end(boolean interrupted){
          m_drivetrain.arcadeDrive(0,0);
       }
}
```

ROBOTIQUE FIRST QUÉBEC

Commandes composées (Command Group)

- La composition de commandes de base permet de générer des commandes plus complexes; on distingue:
  - SequentialCommandGroup;
  - ParallelCommandGroup;
  - ParallelRaceGroup;
  - ParallelDeadGroup.

ROBOTIQUE FIRST QUÉBEC

SequentialCommandGroup

- Dans cette configuration, une liste de commandes est exécutée en série: de la première commande dans la liste à la dernière. La fin de l’exécution de la commande coïncide avec la fin de l’exécution de la dernière commande de la liste.

- Important de s’assurer que les conditions de fin d’exécution de chaque commande soient parfaitement remplies.

ROBOTIQUE FIRST QUÉBEC

ParallelCommandGroup

- Dans cette configuration, les commandes présentes sur la liste de commandes sont exécutées en parallèle. La fin de l’exécution de la commande coïncide avec la fin de l’exécution de toutes les commandes de la liste.

- Important de s’assurer que les conditions de fin d’exécution de chaque commande soient parfaitement remplies.

ROBOTIQUE
FIRST
QUÉBEC

ParallelRaceGroup

- Dans cette configuration, les commandes présentes sur la liste de commandes sont exécutées en parallèle. La fin de l’exécution de la commande est dictée par la première commande qui termine son exécution.

ParallelDeadGroup

- Dans cette configuration, les commandes présentes sur la liste de commandes sont exécutées en parallèle. La fin de l’exécution de la commande a lieu à l’instant où *une commande spécifique,* appelée *Deadline,* termine son exécution.

ROBOTIQUE FIRST QUÉBEC

Ressources

WPILIB Documentation en ligne

Github (WPIlib Suite)

CTRE

Revrobotics

- Conseils pratiques
  - Travail en équipe (choix des capteurs), codage progressif (une fonctionnalité à la fois, tests)
  - Détection des erreurs (erreurs liées au câblage électrique, erreurs software)

ROBOTIQUE FIRST QUÉBEC

Questions

- Contactez nous : rbekale@robotiquefirstquebec.org