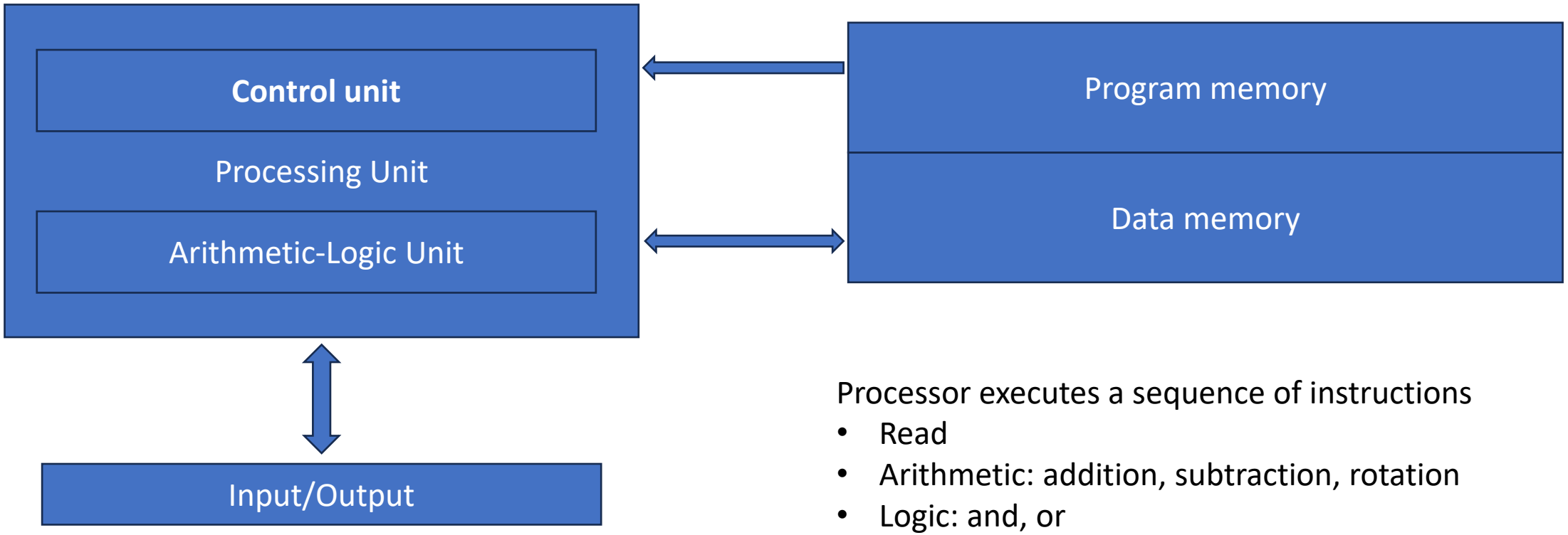
Java for robotics Boot camp

“Von Neumann” architecture



Processor executes a sequence of instructions

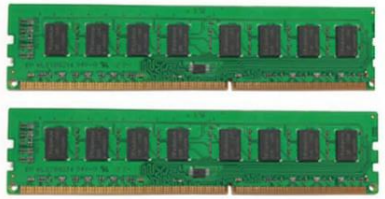
- Read
- Arithmetic: addition, subtraction, rotation
- Logic: and, or
- Write
- Branch, branch if true

Modern Computer architecture

Processor
a.k.a. Central Processing Unit (CPU)
Commercially made by Intel, AMD, ARM, Apple, etc…
Executes instructions from memory

Read Only Memory (ROM)
a.k.a. boot memory, Basic Input/Output System (BIOS), boot flash

Random Access Memory (RAM)
a.k.a. memory, mem
In kilobytes (kB), megabytes (MB), gigabytes (GB)



Input/Output

General Purpose port (GPIO)
Pulse Width Modulation (PMW) or Analog ports
Serial ports (UART, I2C, SPI)
Controller Area Network (CAN bus)

Bus (USB, PCI, PCIe, …)

Keyboard
Mouse

Ethernet
WiFi

Camera
USB stick

"Hard drive"*
or Solid State Drive (SSD)

Graphics card
a.k.a. Graphics Processing Unit (GPU)

*Storage is usually flash memory, measured in GB

Running code

1. Power on or reset the system
2. Processor starts, runs the BIOS (bootloader)
3. Bootloader loads and runs the OS (Operating System), i.e. Windows, Linux, RoboRIO
4. OS initializes all connected systems, connects to the network

---

1. You write the source code in Java using the WPILib environment in Visual Studio Code (or C++, Python, C#, …)
2. Compiler creates machine code (or intermediate code) from the source code and libraries
3. Loader deploys and runs the program/application

Java concepts

- Programming language (for source code)
- Runs on many platforms (PC, phone, server, …)
- Object Oriented Programming
- It is NOT JavaScript

Tutorial at https://www.w3schools.com/java/default.asp

Java syntax

```
// This is a comment
public class Example {
  public static void main(String[] args) {
    System.out.println("Hello World");
  }
}
```

- Comments are included in the code; explain what you are doing
- All code is contained in classes
- The class name is PascalCase (first letter of each word capitalized)
- The file name must match the class name (Example.java)
- Curly braces {} delimit a block of code
- Semicolon ; ends a code statement
- The main() method is where your program starts

Java variables

```
int myNum = 5; // Integer (whole number)
float myFloatNum = 5.99f; // Floating point number
char myLetter = 'D'; // Character
boolean myBool = true; // Boolean
String myText = "Hello"; // String

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
int[] myNum = {10, 20, 30};

System.out.println(cars[0]);
// Outputs Volvo

System.out.println(cars.length);
// Outputs 4

System.out.println(myNum[2]);
// Outputs 30

System.out.println(cars[3]);
// Outputs???
```

- int - stores integers, without decimals, such as 123 or -123
- float - stores numbers with decimals, such as 19.99 or -19.99
- char - stores single characters, such as 'a' or 'B'
- boolean - stores values with two states: true or false
- String - stores text, such as "Hello“

- Variable names (identifiers) are camelCase (first letter lower case, then first letter of each word capitalized)
- To create a variable, declare its type and assign a value
  type name = value;
- You can use a variable by its identifier ***int other = myNum;*** or assign it another value ***myNum=10;***

- Arrays contain multiple elements of the same type
- Access by index, starts at 0
- Length is the size of the array

Java flow control

```java
int x = 20;
int y = 18;
if (x > y) {
  System.out.println("x is greater than y");
} else {
  System.out.println("y is greater than x");
}

int z = 0;
while (z < 5) {
  System.out.println(z);
  z++;
}

// Outer loop
for (int i = 1; i <= 2; i++) {
  System.out.println("Outer: " + i); // Executes 2
times
  // Inner loop
  for (int j = 1; j <= 3; j++) {
    System.out.println(" Inner: " + j); // Executes
6 times (2 * 3)
  }
}
```

- ***if() {} else {}*** specifies a block of code to be executed if a condition is true (else is optional)
- ***while() {}*** specifies a block of code to be executed as long as a condition is true
- ***for(before;condition;iteration) {}*** specifies a block of code to be executed as long as a condition is true
- Loops and ifs may be nested

- ***break;*** jumps out of a loop
- ***continue;*** jumps to the next iteration of a loop

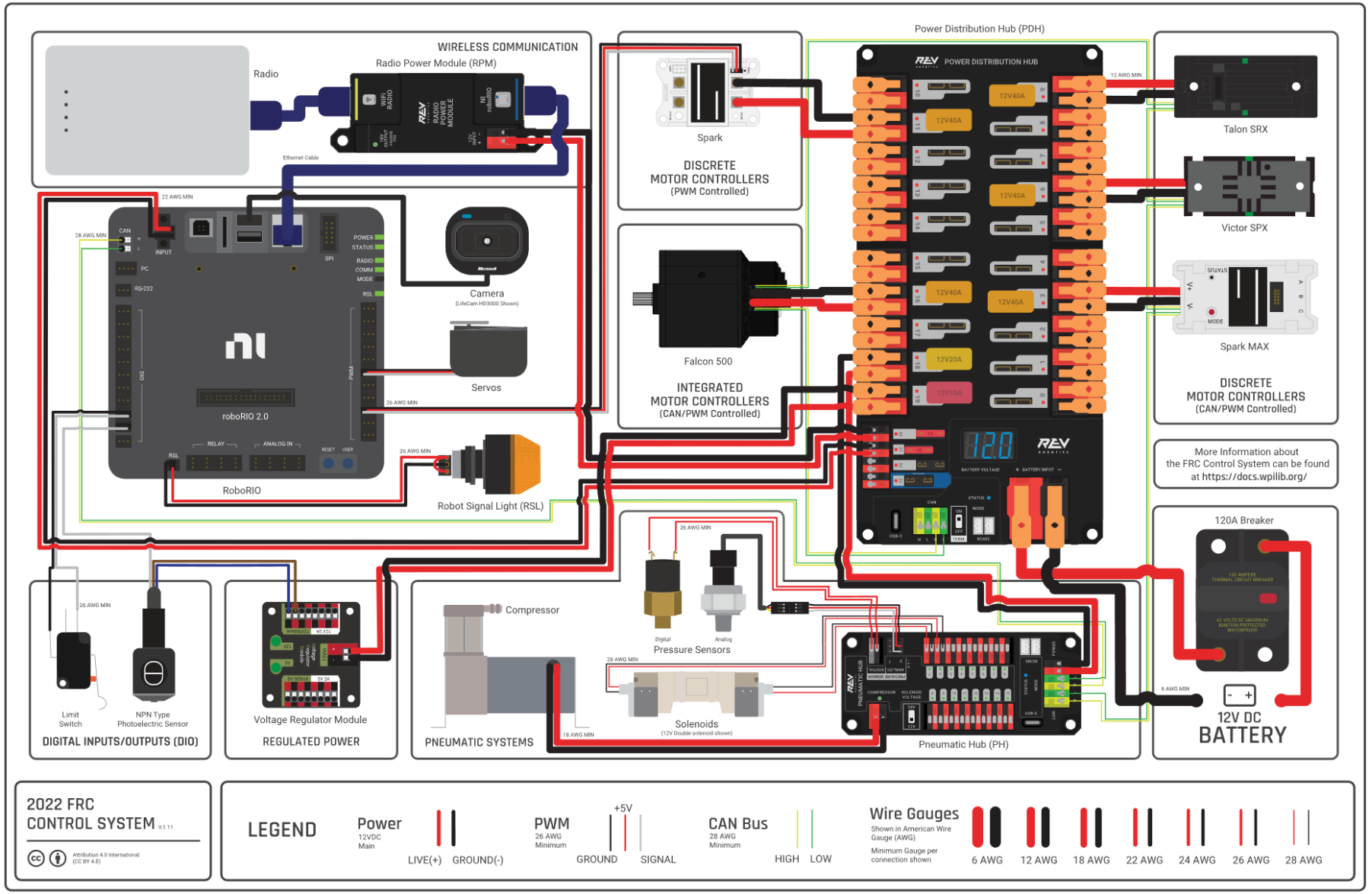
Object Oriented Programming (classes and objects)

- How do we break down a complex problem in small, independent, easy to solve problems. Abstractions and contracts.
- A class represent a concept or template
  - class Fruit {}
  - class Car {}
- An object is a specific instance of a class
  - Fruit myFruit = new Fruit(“banana”);
  - Car myCar = new Car(“Tesla”);
- Classes define attributes and methods
  - myFruit.m_name = “banana”;
  - myFruit.m_family = “Musaceae”;
  - myFruit.Eat();
  - myCar.StartEngine();
- Objects are instantiated using constructors
  - class Age { float m_age; public Age(int years, int days) { m_age = years + (float)days / 365; } }
  - Age myAge = new Age(18,175);
- A class may inherit from another class: reuse the attributes and methods of the parent class
  - class Plant { void Grow() {} }
  - class Tree extends Plant {}
  - class EvergreenTree extends Tree {}

Using Git

- **Repository/repo**: your files, folder structure and change history; stored on the github server, i.e. at https://github.com/RamenRobotics9036/RamenBot2024
- **Clone**: to make a copy of the server repo on your machine, i.e. a local repo
- **Commit**: a particular saved state of the repo, i.e. a snapshot of all files; to commit is to create that snapshot, along with an explanation for the changes
- **Branch**: a means of grouping a set of commits with their history. For example:
  - the *`main`* branch may contain all tested code
  - the *`release/june2023`* branch may contain last year's working code
  - the *`user/guillaume`* branch may contain all changes this contributor is currently working on
  - the *`feature/swerve`* branch may contain all changes currently related to the swerve feature
- **Push**: update the server repo with your changes
- **Pull**: update your local repo with changes from the server repo
- **Merge**: combine various changes from different branches/commits
- **Pull request**: a proposal to merge changes between two branches, usually to *`main`*
  It allows collaborators to review and discuss the changes before they are integrated

RoboRIO


WIRELESS COMMUNICATION
Radio
Radio Power Module (RPM)
Ethernet Cable
22 AWG MIN
28 AWG MIN
CAN
INPUT
PC
RS-232
SPI
POWER
STATUS
RADIO
COMM
MODE
RSL
DIO
PWM
roboRIO 2.0
RELAY
ANALOG IN
RESET
USER
RoboRIO
26 AWG MIN
Camera
(LifeCam HD3000 Shown)
Servos
Robot Signal Light (RSL)
Spark
DISCRETE
MOTOR CONTROLLERS
(PWM Controlled)
Falcon 500
INTEGRATED
MOTOR CONTROLLERS
(CAN/PWM Controlled)
Power Distribution Hub (PDH)
POWER DISTRIBUTION HUB
12V40A
12V20A
12V10A
12 AWG MIN
Talon SRX
Victor SPX
Spark MAX
DISCRETE
MOTOR CONTROLLERS
(CAN/PWM Controlled)
More Information about
the FRC Control System can be found
at https://docs.wpilib.org/
120A Breaker
6 AWG MIN
12V DC
BATTERY
Limit
Switch
NPN Type
Photoelectric Sensor
DIGITAL INPUTS/OUTPUTS (DIO)
Voltage Regulator Module
REGULATED POWER
Compressor
Digital
Analog
Pressure Sensors
Solenoids
(12V Double solenoid shown)
18 AWG MIN
PNEUMATIC SYSTEMS
Pneumatic Hub (PH)
2022 FRC
CONTROL SYSTEM
Attribution 4.0 International
(CC BY 4.0)
LEGEND
Power
12VDC
Main
LIVE(+)
GROUND(-)
PWM
26 AWG
Minimum
+5V
GROUND
SIGNAL
CAN Bus
28 AWG
Minimum
HIGH
LOW
Wire Gauges
Shown in American Wire
Gauge (AWG)
Minimum Gauge per
connection shown
6 AWG
12 AWG
18 AWG
22 AWG
24 AWG
26 AWG
28 AWG

WPILib https://wpilib.org/

- [need content]

Git cheat sheet

https://github.com/RamenRobotics9036/RamenBot2024

Git and code changes

From Visual Studio

1. Switch to `main` branch
2. Pull (copy newest changes from github)
3. Create a new branch based on `main`, call it something related to your name or the task (no spaces), for example `visiontest` or
4. Code, deploy, test, repeat
5. Stage, review, and submit your changes (with a comment)
6. Push (copy your branch updates to github)

All unsubmitted changes will be lost!

Create a pull request when complete.

Resuming work in progress

From Visual Studio

1. Switch to `yourbranchname` branch
2. Pull (copy newest changes from github)
3. Code, deploy, test, repeat
4. Stage, review, and submit your changes (with a comment)
5. Push (copy your branch updates to github)

All unsubmitted changes will be lost!

Create a pull request when complete.

Create a pull request

Once code changes are completed and ready for everybody to use.

From https://github.com/RamenRobotics9036/RamenBot2024

1. Create a pull request base:`main` compare:`yourbranchname`
2. Reviewer looks at the code, may suggest changes
3. If changes needed, follow “resuming work in progress”
4. Once approved, github copies/merges the code to main