

Introdução ao Arduino: Comunicação Java-Arduino

Vinicius Souza de Jesus ¹
Heder Dorneles Soares ²
Carlos Eduardo Pantoja ^{1,2}
Jose Viterbo ²

- 1. Centro Federal de Educação Tecnológica (CEFET/RJ), Brasil
- 2. Universidade Federal Fluminense (UFF), Brasil

Disciplina de Computação
Ubíqua

4 de Setembro de 2018



OUTLINE

1. Introdução

2. Usando o Arduino

3. Javino

4. Exemplos

5. Conclusão

Referências Bibliográficas

OUTLINE

1. Introdução

2. Usando o Arduino

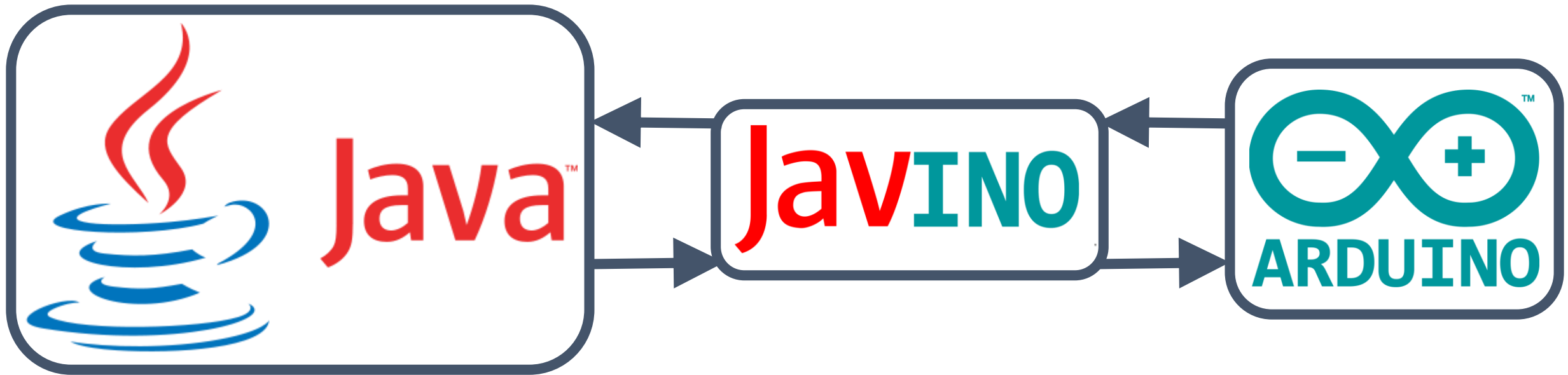
3. Javino

4. Exemplos

5. Conclusão

Referências Bibliográficas

1. INTRODUÇÃO



1. INTRODUÇÃO: OBJETIVOS

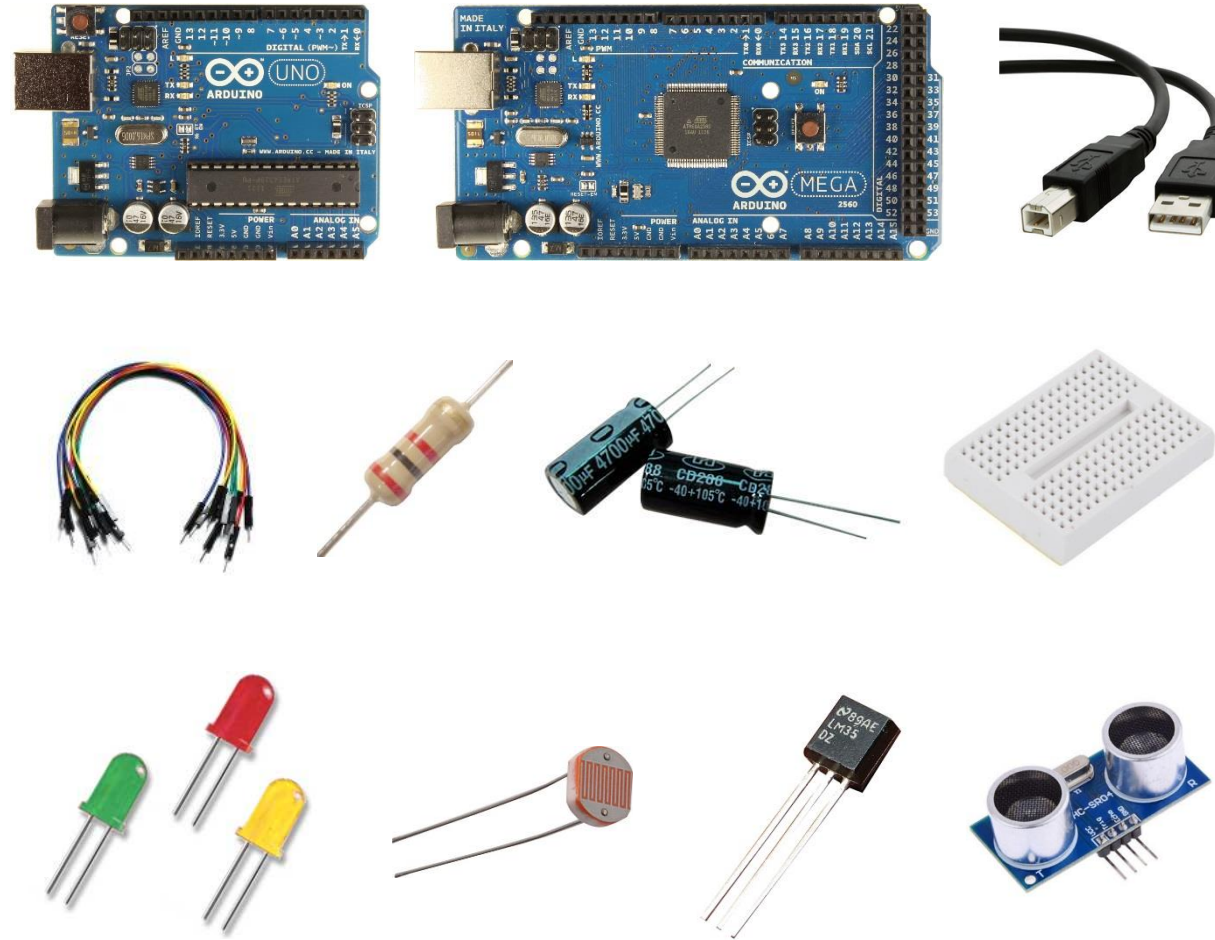
Objetivo Principal

- Expandir, para fins estudantis e de versatilidade, a plataforma de comunicação, de forma a otimizar a integridade e o contexto das mensagens enviadas ou recebidas, possibilitando também a recepção de conteúdos, não só pelo java, mas também pelo arduino.

Objetivo Secundário

- Introduzir conceitos básicos de Arduino
- Introduzir conceitos básicos do Javino

1. INTRODUÇÃO: KIT ARGO-JAVINO



OUTLINE

1. Introdução

2. Usando o Arduino

3. Javino

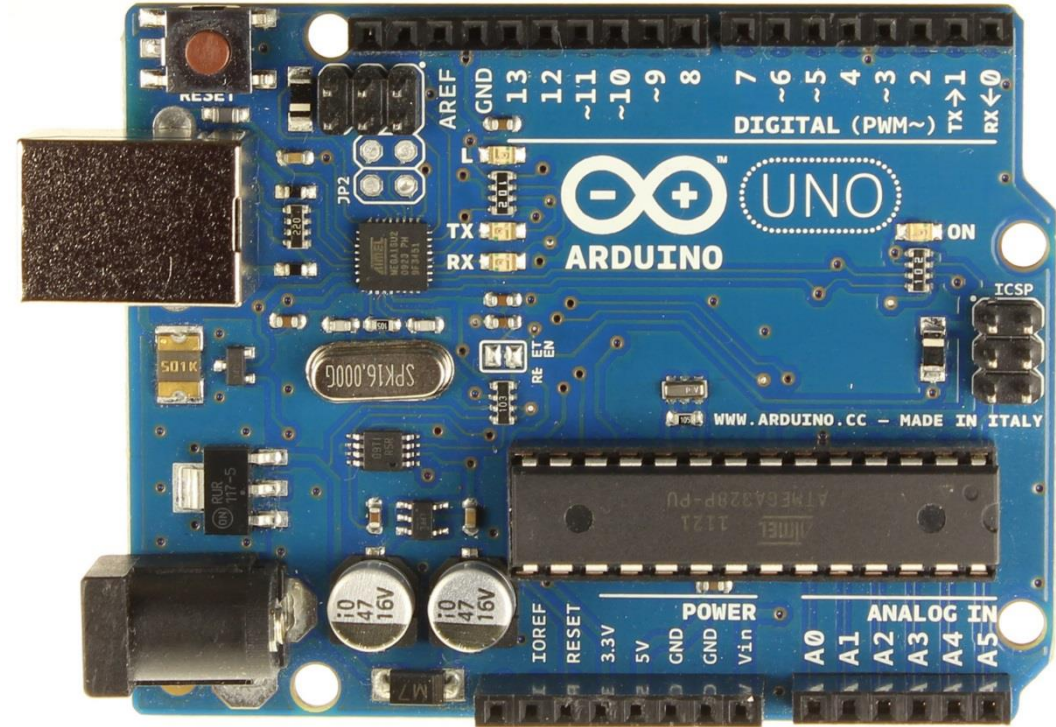
4. Exemplos

5. Conclusão

Referências Bibliográficas

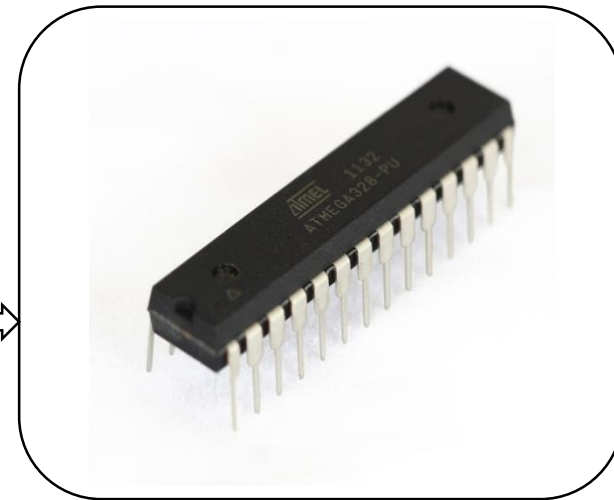
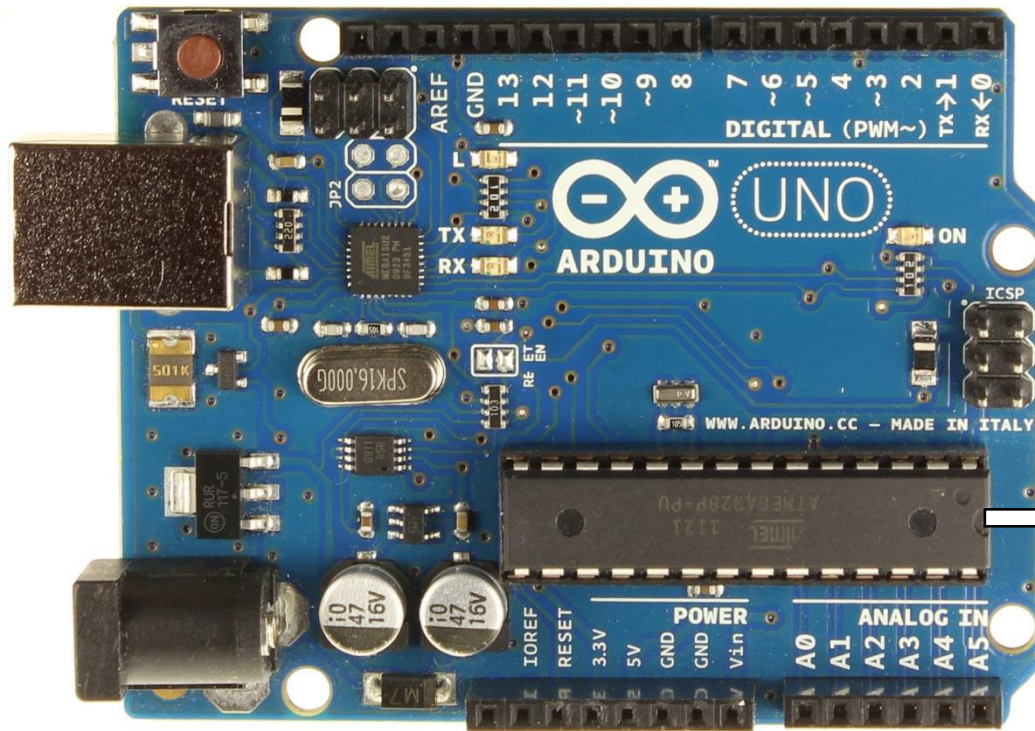
2. ARDUINO

O Arduino é um componente que une conceitos principalmente de **eletrônica** e **programação**, a fim de facilitar a aplicação de **projetos tecnológicos**.



2. ARDUINO

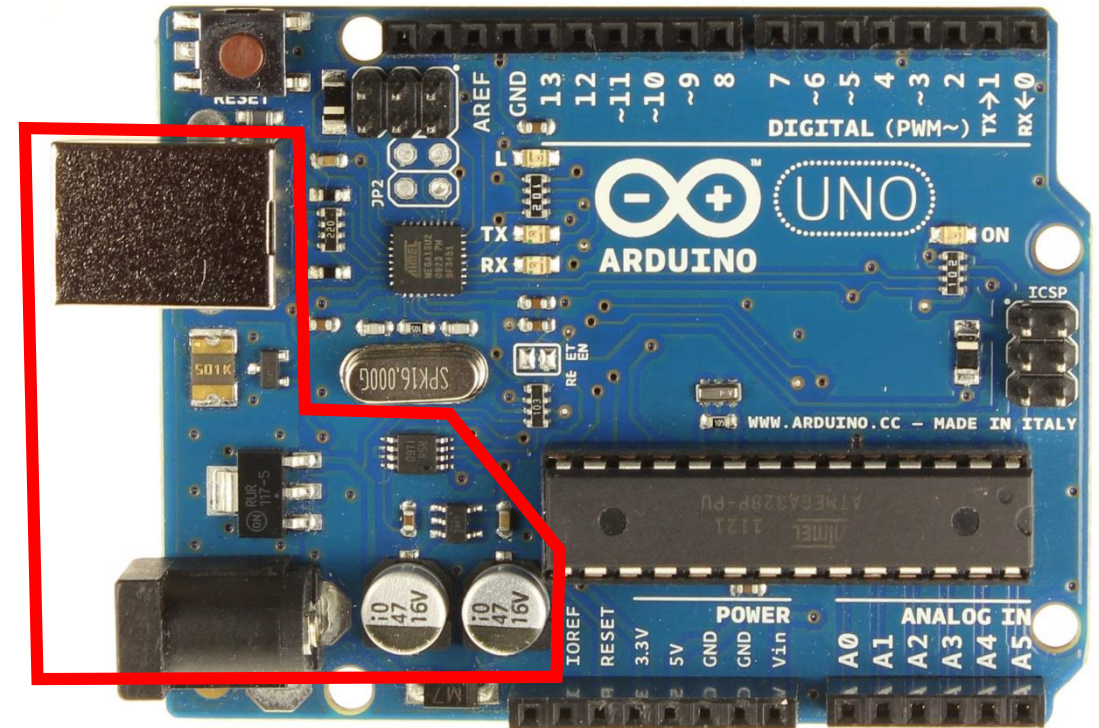
Seu componente principal é o **ATMEGA**, um micro controlador que atua como **cérebro** do projeto a ser implementado.



2. ARDUINO

Sua parte elétrica possui todas as aplicações necessárias para o funcionamento e utilização dos recursos do ATMEGA. Suas aplicações mais importantes são:

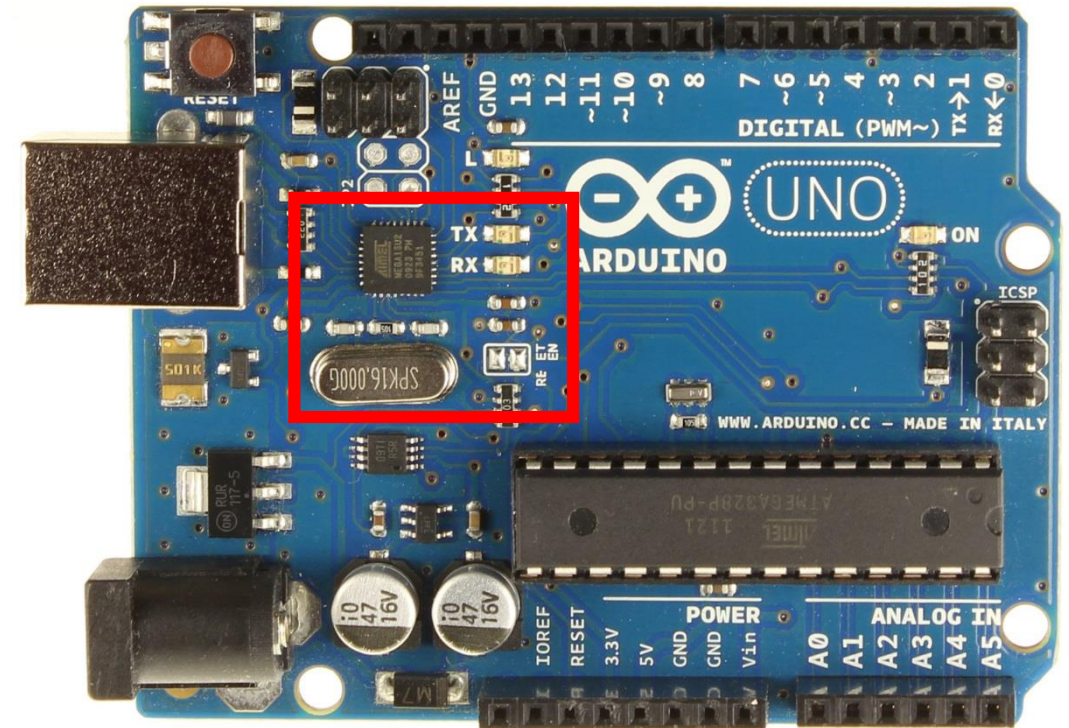
- 1- Filtragem de alimentação;**
- 2- Conversão de sinal serial para USB;
- 3- Regulagem de frequência;
- 4- Botão Reset;
- 5- Conectores de sinais de entrada ou saída;
- 6- Conectores de alimentação;



2. ARDUINO

Sua parte elétrica possui todas as aplicações necessárias para o funcionamento e utilização dos recursos do ATMEGA. Suas aplicações mais importantes são:

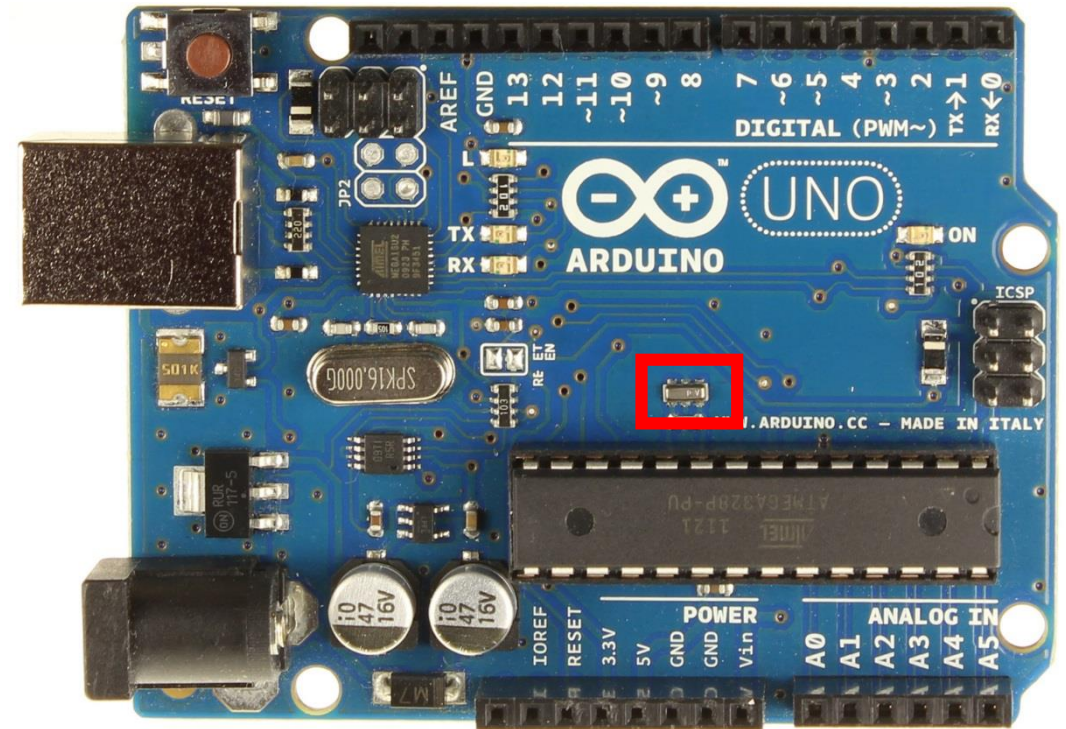
- 1- Filtragem de alimentação;
- 2- Conversão de sinal serial para USB;**
- 3- Regulagem de frequência;
- 4- Botão Reset;
- 5- Conectores de sinais de entrada ou saída;
- 6- Conectores de alimentação;



2. ARDUINO

Sua parte elétrica possui todas as aplicações necessárias para o funcionamento e utilização dos recursos do ATMEGA. Suas aplicações mais importantes são:

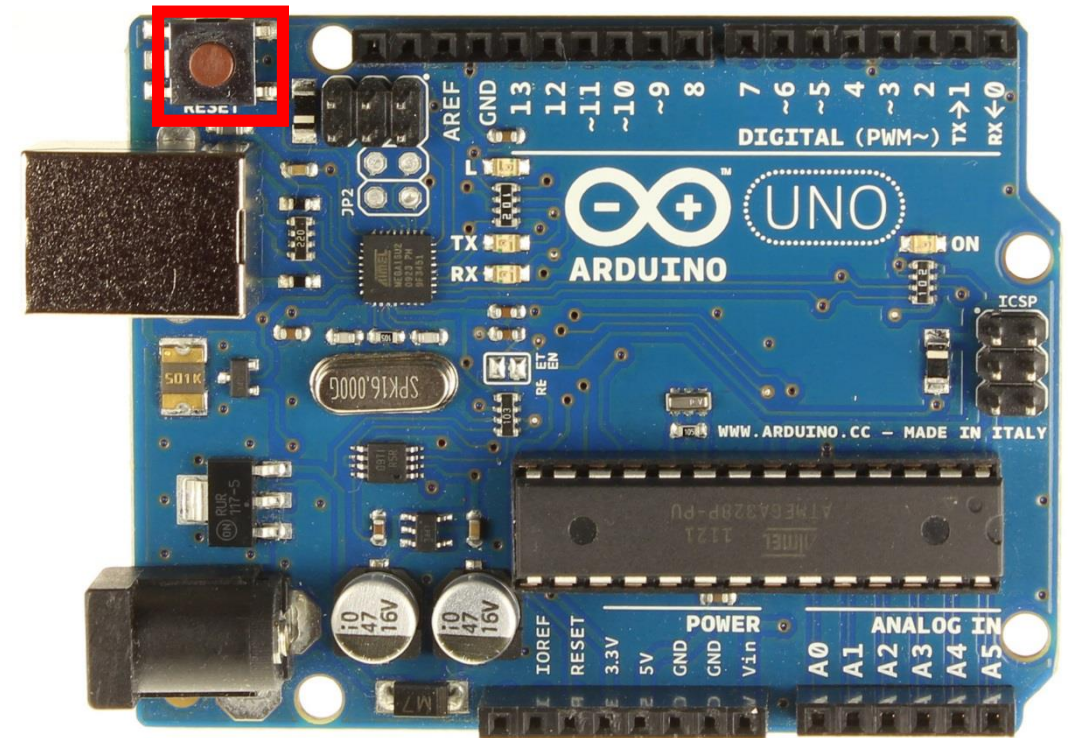
- 1- Filtragem de alimentação;
- 2- Conversão de sinal serial para USB;
- 3- Regulagem de frequência;**
- 4- Botão Reset;
- 5- Conectores de sinais de entrada ou saída;
- 6- Conectores de alimentação;



2. ARDUINO

Sua parte elétrica possui todas as aplicações necessárias para o funcionamento e utilização dos recursos do ATMEGA. Suas aplicações mais importantes são:

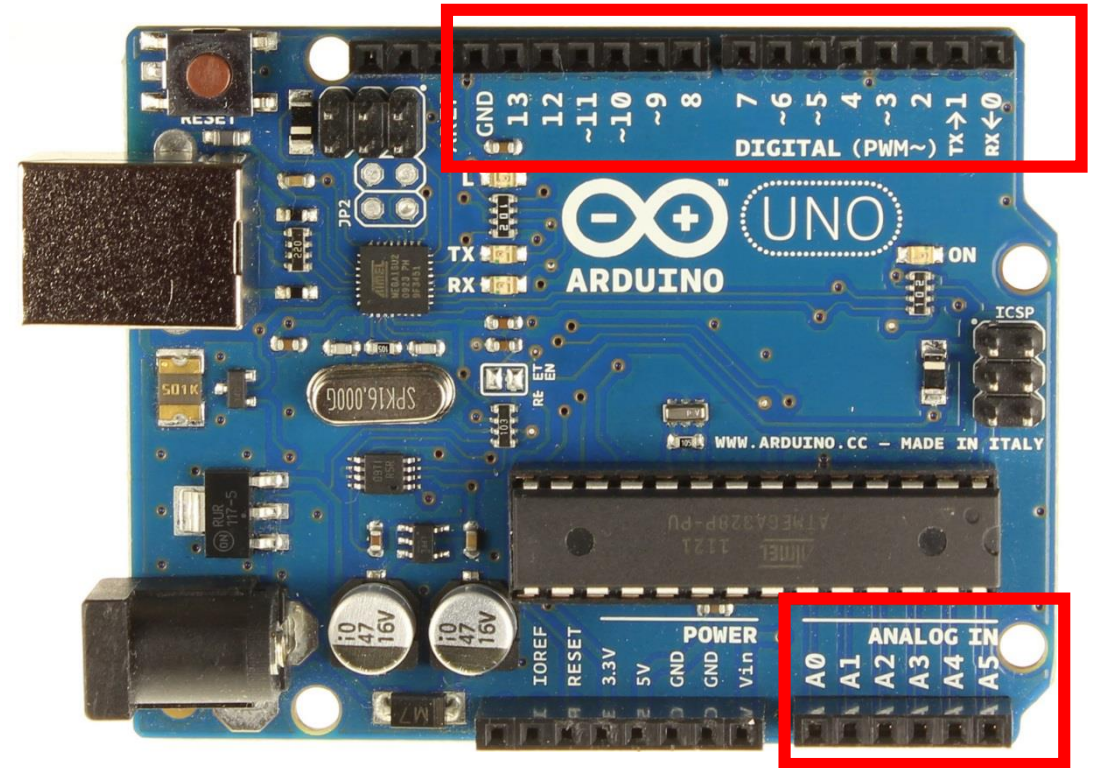
- 1- Filtragem de alimentação;
- 2- Conversão de sinal serial para USB;
- 3- Regulagem de frequência;
- 4- Botão Reset;**
- 5- Conectores de sinais de entrada ou saída;
- 6- Conectores de alimentação;



2. ARDUINO

Sua parte elétrica possui todas as aplicações necessárias para o funcionamento e utilização dos recursos do ATMEGA. Suas aplicações mais importantes são:

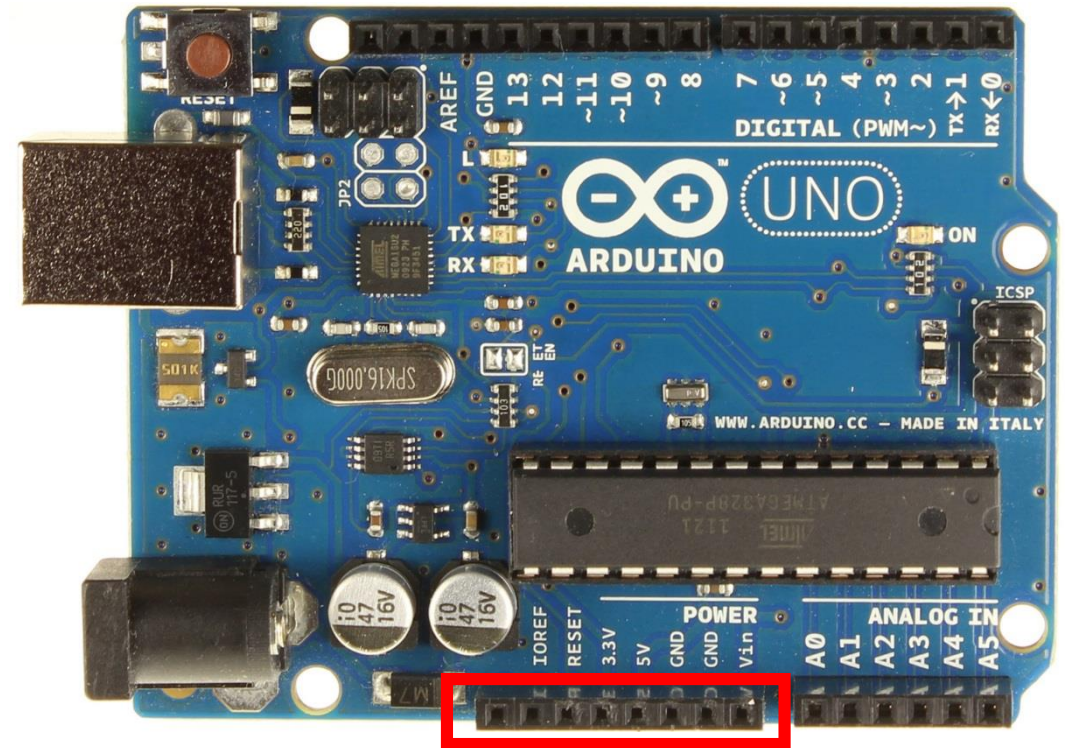
- 1- Filtragem de alimentação;
- 2- Conversão de sinal serial para USB;
- 3- Regulagem de frequência;
- 4- Botão Reset;
- 5- Conectores de sinais de entrada ou saída;**
- 6- Conectores de alimentação;



2. ARDUINO

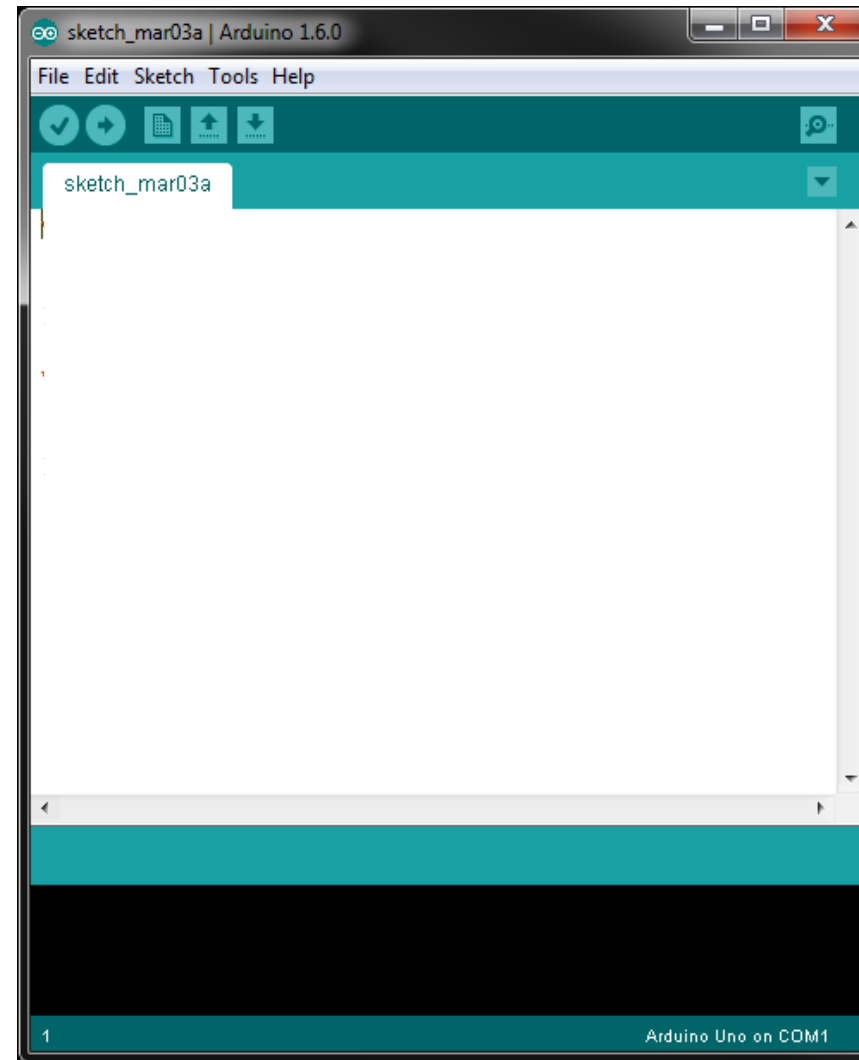
Sua parte elétrica possui todas as aplicações necessárias para o funcionamento e utilização dos recursos do ATMEGA. Suas aplicações mais importantes são:

- 1- Filtragem de alimentação;
- 2- Conversão de sinal serial para USB;
- 3- Regulagem de frequência;
- 4- Botão Reset;
- 5- Conectores de sinais de entrada ou saída;
- 6- Conectores de alimentação.**



2. SOFTWARE DO ARDUINO

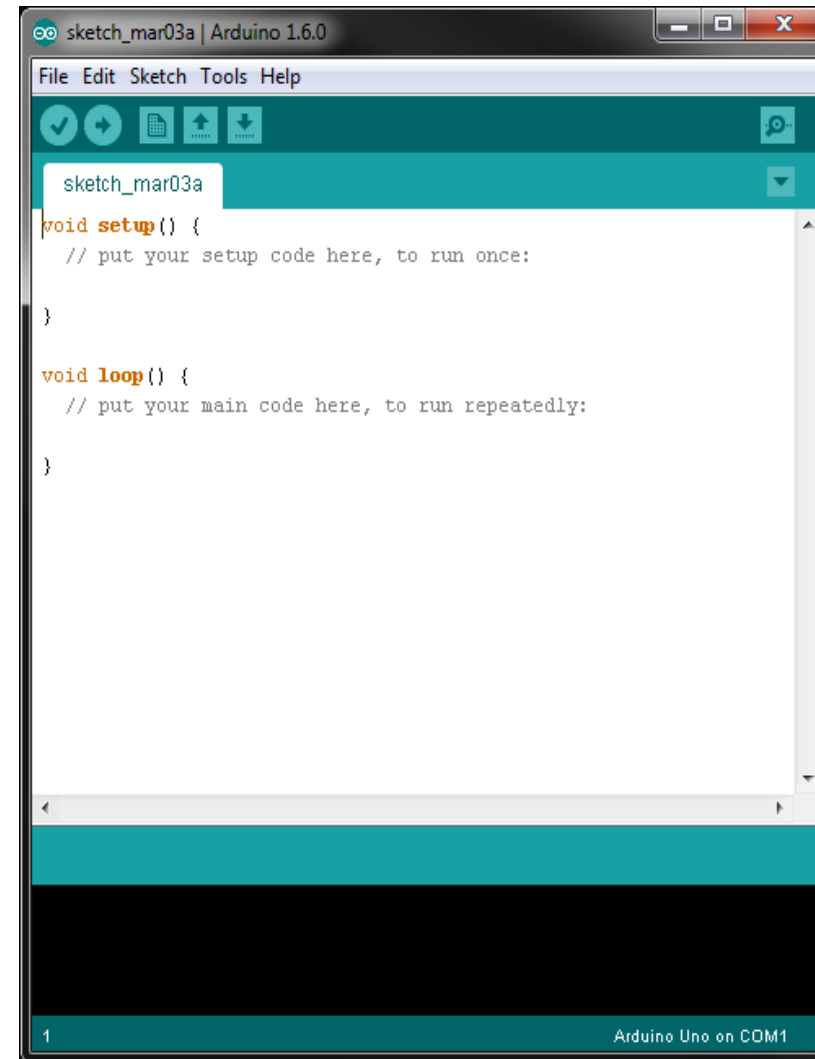
Seu software é conhecido
como **Arduino IDE** e é
baseado na **linguagem C**.



2. SOFTWARE DO ARDUINO

Este possui duas funções obrigatórias:

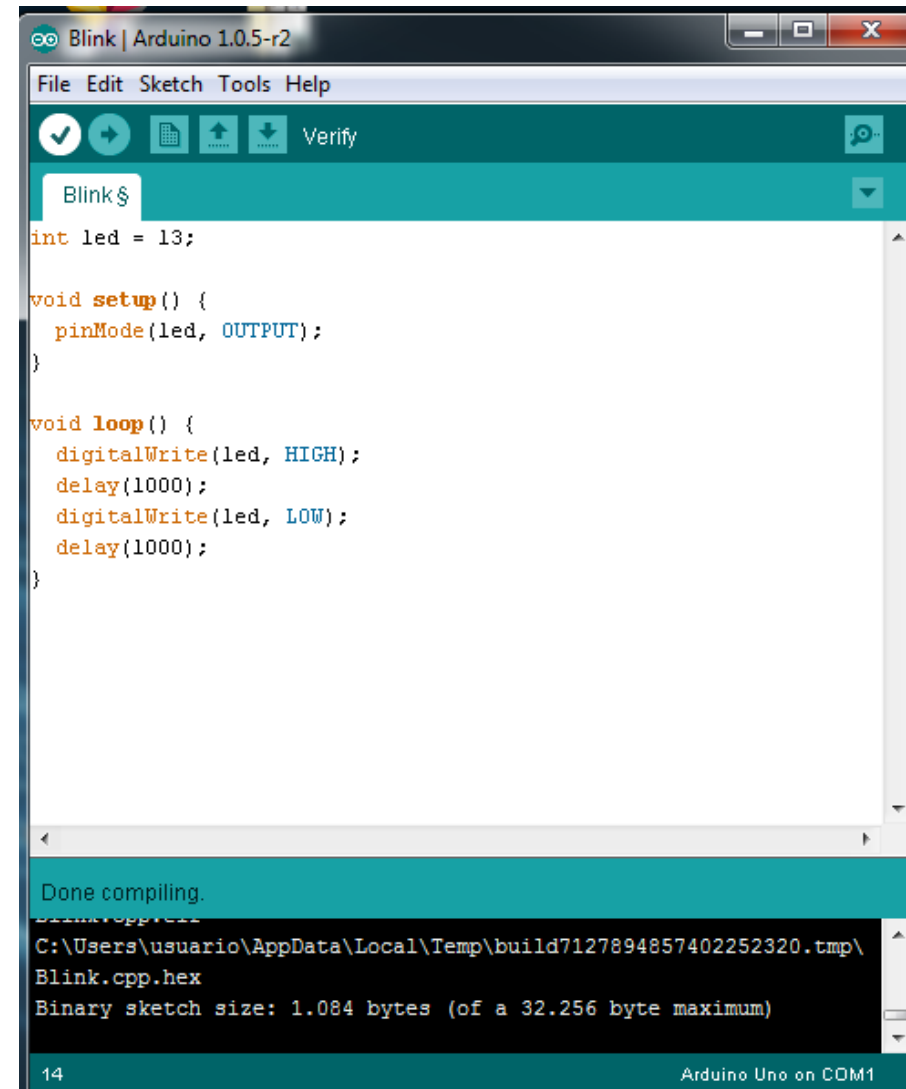
- 1- **setup**: configura funcionalidades de **pré-iniciação**;
- 2- **loop**: **repetição infinita** do código escrito;



2. ARDUINO: CRIANDO UM PROJETO

Codificando...

Escreva o seguinte código:



The screenshot shows the Arduino IDE interface. The main window displays the 'Blink' sketch code. The code defines a pin (led = 13) and a loop that turns the LED on for 1000ms and off for 1000ms. The bottom panel shows the serial monitor with the message 'Done compiling.' and the file path 'C:\Users\usuario\AppData\Local\Temp\build7127894857402252320.tmp\Blink.cpp.hex'. It also indicates the binary sketch size is 1.084 bytes.

```
int led = 13;

void setup() {
  pinMode(led, OUTPUT);
}

void loop() {
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}
```

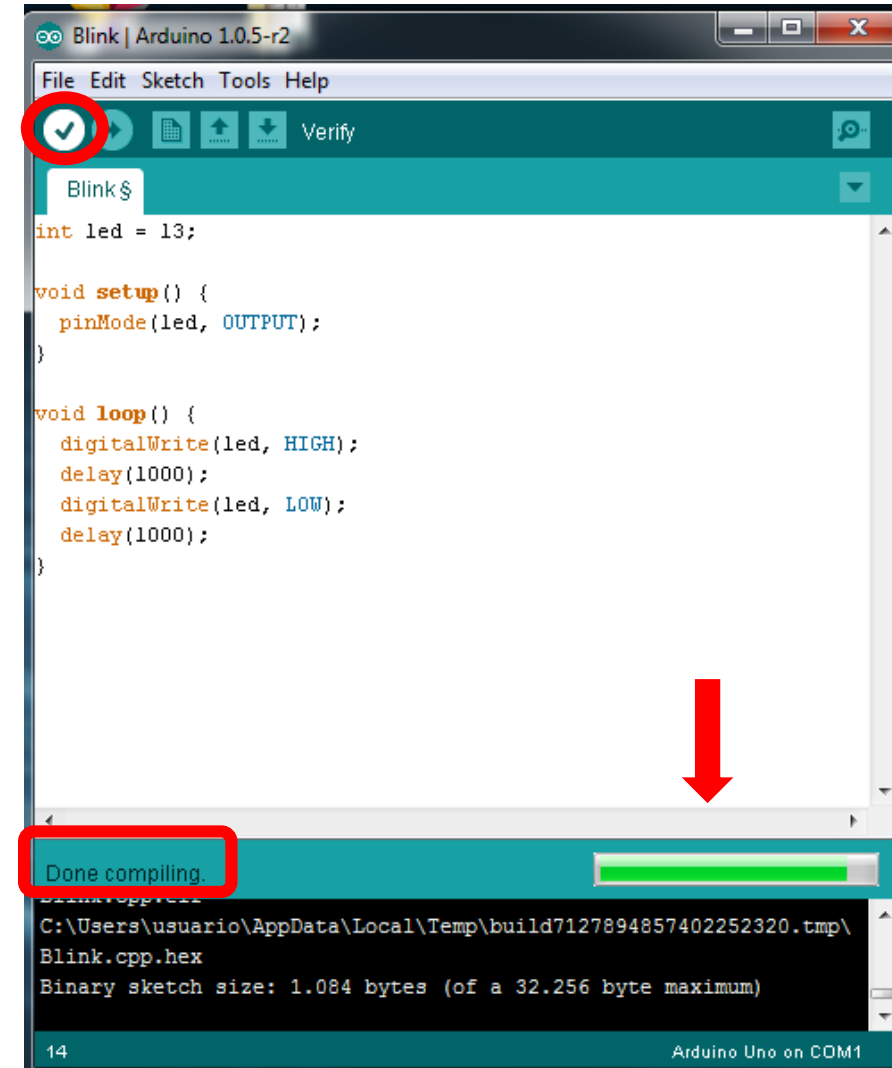
Done compiling.
C:\Users\usuario\AppData\Local\Temp\build7127894857402252320.tmp\Blink.cpp.hex
Binary sketch size: 1.084 bytes (of a 32.256 byte maximum)

14 Arduino Uno on COM1

2. ARDUINO: CRIANDO UM PROJETO

Compilando...

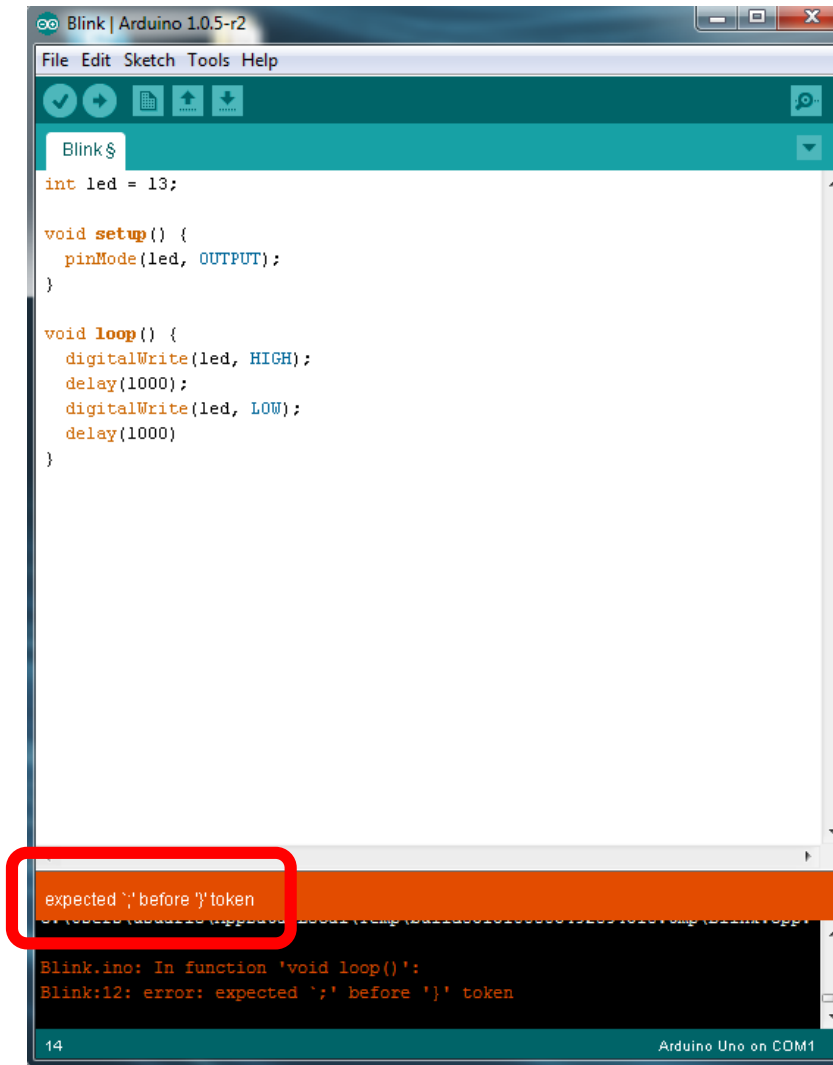
Compile no botão destacado.
Este botão irá verificar se o
programa contém **erros** de
sintaxe;



2. ARDUINO: CRIANDO UM PROJETO

Compilando...

Erros de compilação são exibidos na área inferior da IDE do Arduino, indicando a causa do problema. No exemplo abaixo, foi identificado a falta de ponto e vírgula como erro;



The screenshot shows the Arduino IDE interface with a sketch named 'Blink'. The code in the main editor is as follows:

```
int led = 13;

void setup() {
  pinMode(led, OUTPUT);
}

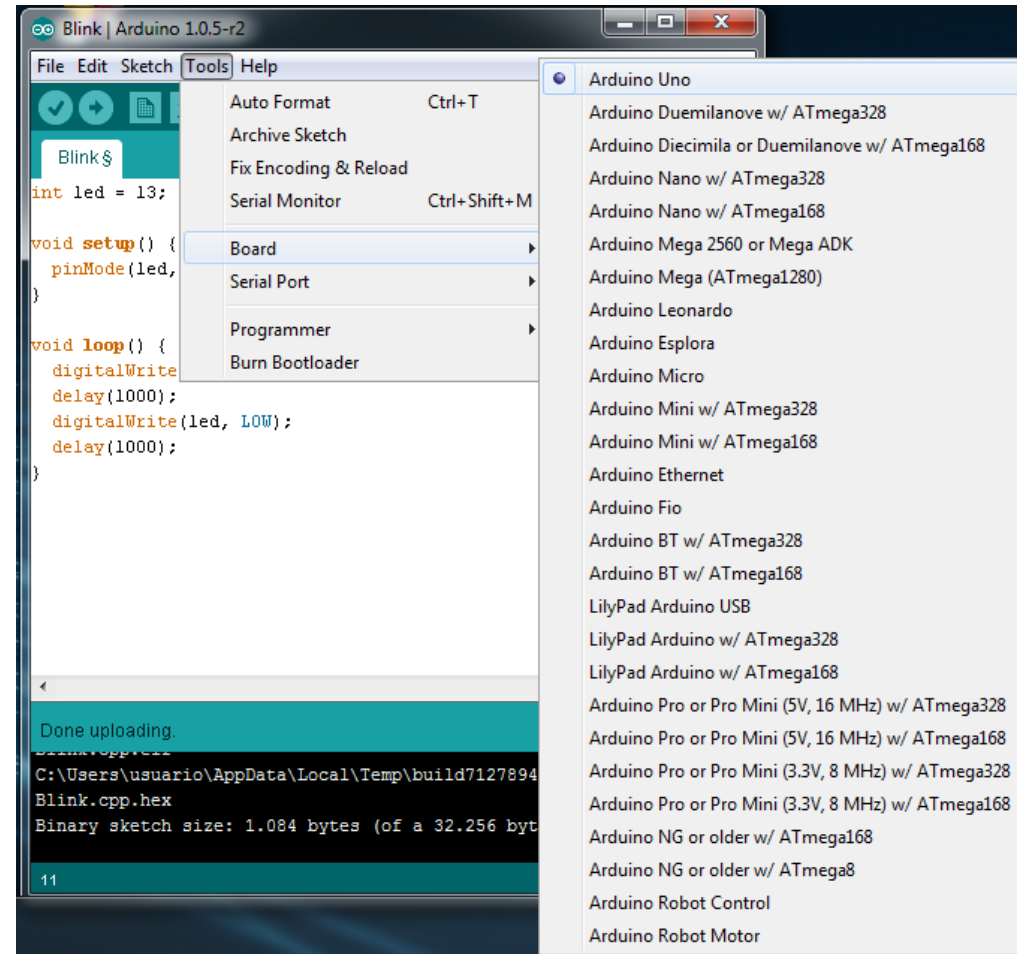
void loop() {
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000)
}
```

The error message, highlighted with a red box, is: "expected ';' before ')' token". Below the error message, the IDE shows the file path "Blink.ino: In function 'void loop()':" and the specific error: "Blink:12: error: expected ';' before ')' token". The status bar at the bottom indicates "14" and "Arduino Uno on COM1".

2. ARDUINO: CRIANDO UM PROJETO

Ajustar...

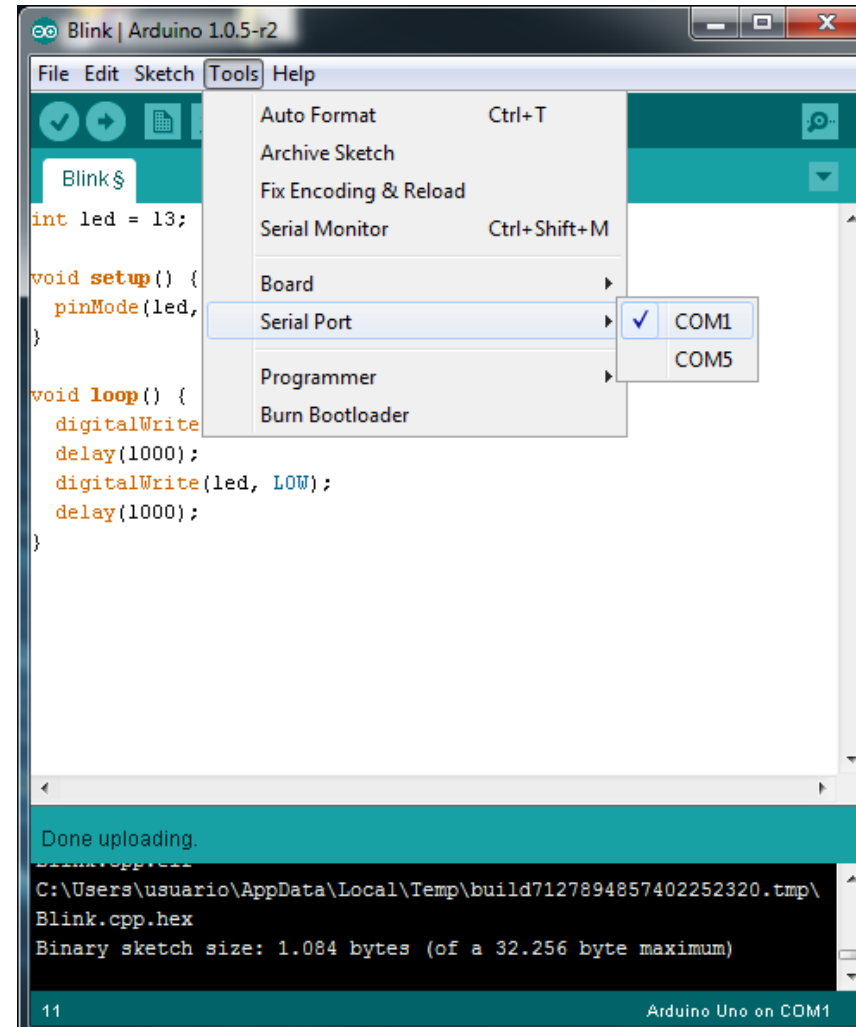
Agora verifique se seu software entregará o programa ao Arduino correto. Primeiramente, vá em **Tools->Board** e escolha a **versão** da placa utilizada;



2. ARDUINO: CRIANDO UM PROJETO


Ajustar...

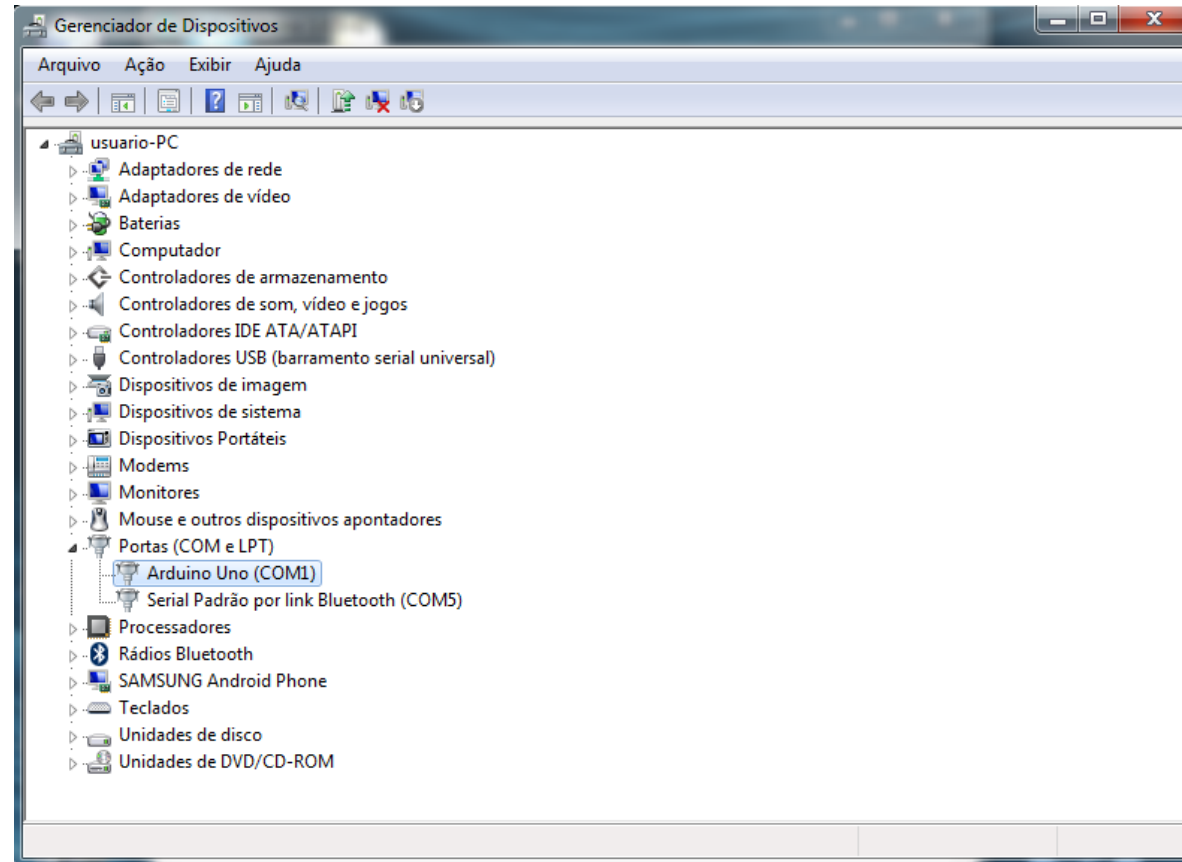
Depois, vá em **Tools->Serial Port** e escolha a porta **COM** que o computador está utilizando para estabelecer a comunicação com o Arduino;



2. ARDUINO: CRIANDO UM PROJETO

Ajustar...

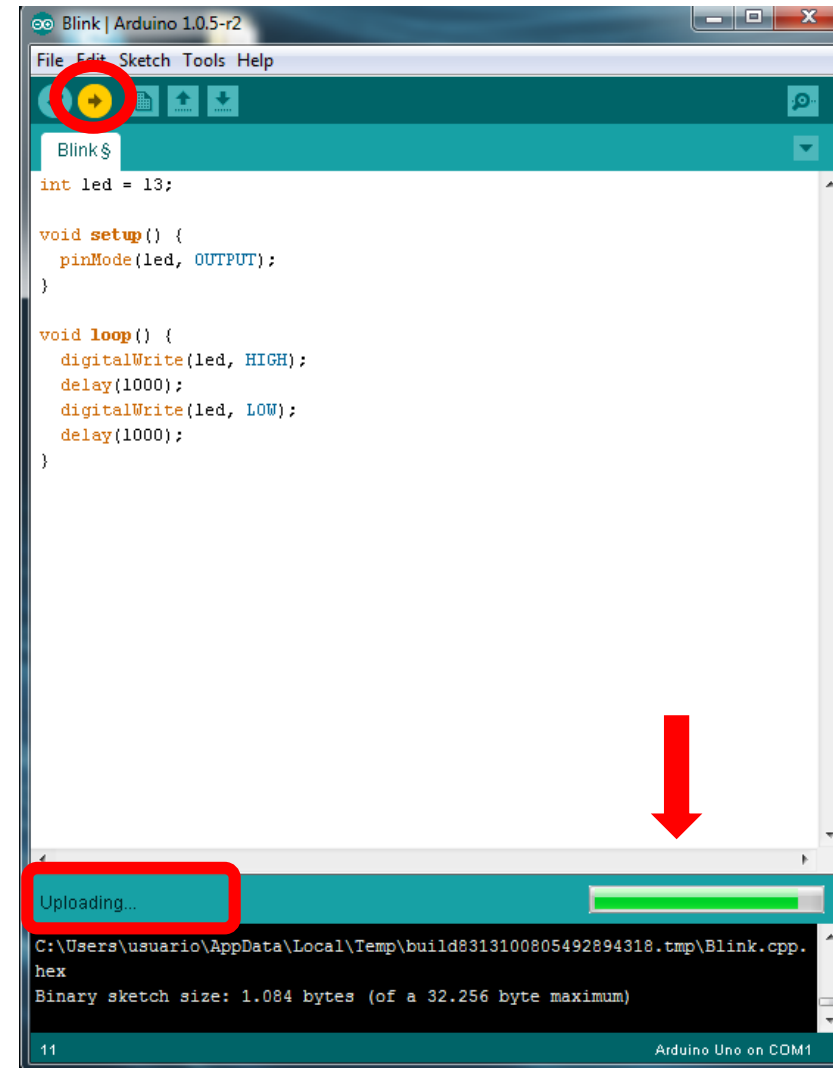
Em caso de não conhecer a porta de comunicação, vá na barra de pesquisa do Windows e digite "[gerenciador de dispositivos](#)", ou apenas realize o atalho  + "[pause](#)", clique em **portas (COM e LPT)** e certifique-se do local do driver do Arduino;



2. ARDUINO: CRIANDO UM PROJETO

Carregar...

Caso a sintaxe esteja correta, clique no botão destacado para **carregar** o programa para a memória do Arduino. **Erros** durante o carregamento do programa na placa também são indicados no campo de status;



2. ARDUINO: PROGRAMANDO

Na função **setup**, o programador **configura parâmetros** necessários para iniciar a rotina de loop. Por exemplo, os pinos de comando devem ser definidas como entrada ou saída de dados para que o ATMEGA saiba se deve escrever ou ler informações;

```
void setup() {  
    pinMode(13, OUTPUT);  
    pinMode(12, INPUT);  
    pinMode(A0, INPUT);  
}
```

2. ARDUINO: PROGRAMANDO

Na função **loop**, o programador faz rotinas de **execuções infinitas**, ou seja, ao chegar no final da rotina, o sistema volta ao início da função loop;

```
int cont = 0;  
void loop() {  
    cont++;  
}
```

2. ARDUINO: PROGRAMANDO

Algumas **funções** na **programação de microcontroladores** são essenciais para realizar o acionamento e leitura de pinos, comunicações entre dispositivos, etc. Por exemplo, o arduino permite a **escrita** e a **leitura digital** e a **leitura analógica**. Por isso, existem os comandos `digitalWrite(pino, valor)`, `digitalRead(pino)` e `analogRead(pino)`;

```
void loop() {  
    digitalWrite(13, HIGH);  
    delay(1000);  
    digitalWrite(13, LOW);  
    delay(1000);  
    int valorDoPino12;  
    valorDoPino12 = analogRead(12);  
}
```

2. ARDUINO: PROGRAMANDO

Comunicação Serial:

Serial.begin(velocidade);



SETUP

Serial.available();

Serial.read();

Serial.readString();

Serial.println();



LOOP

2. ARDUINO: PROGRAMANDO

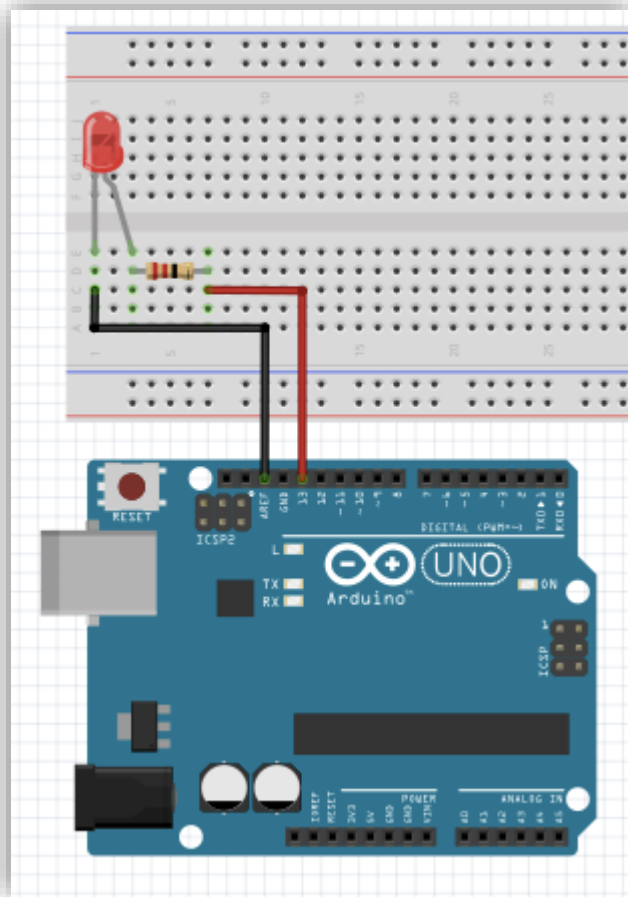
Criando **funções ...**

```
String getLight() {  
    light = analogRead(LDR);  
    return "light("+String(light)+")";  
}  
  
String getTemp() {  
    tempSensor = (analogRead(LM35)*10)*0.00488759;  
    char tempString[10];  
    dtostrf(tempSensor,3,2,tempString);  
    return "temperature("+String(tempString)+")"; }
```

e **procedimentos**

```
void coldSignal() {  
    digitalWrite (ledRed,LOW);  
    digitalWrite (ledBlue,HIGH); }  
  
void coolSignal() {  
    digitalWrite (ledRed,LOW);  
    digitalWrite (ledBlue,LOW); }
```

2. ARDUINO: PROJETO BLINK LED



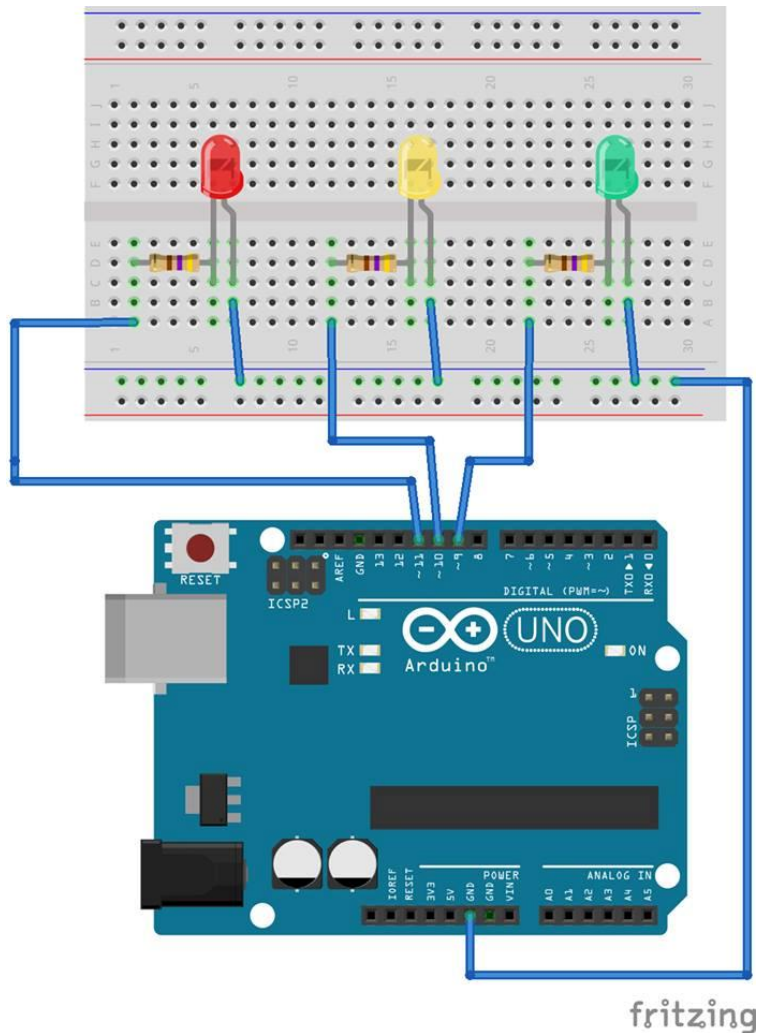
```
Blink | Arduino 1.0.5-r2
File Edit Sketch Tools Help
Blink $
int led = 13;

void setup() {
  pinMode(led, OUTPUT); //Configuração do pino 13 como saída
}

void loop() {
  digitalWrite(led, HIGH); //Liga o pino 13
  delay(1000); //tempo de um segundo
  digitalWrite(led, LOW); //Desliga o pino 13
  delay(1000); //tempo de um segundo.
}
```

11 Arduino Uno on COM1

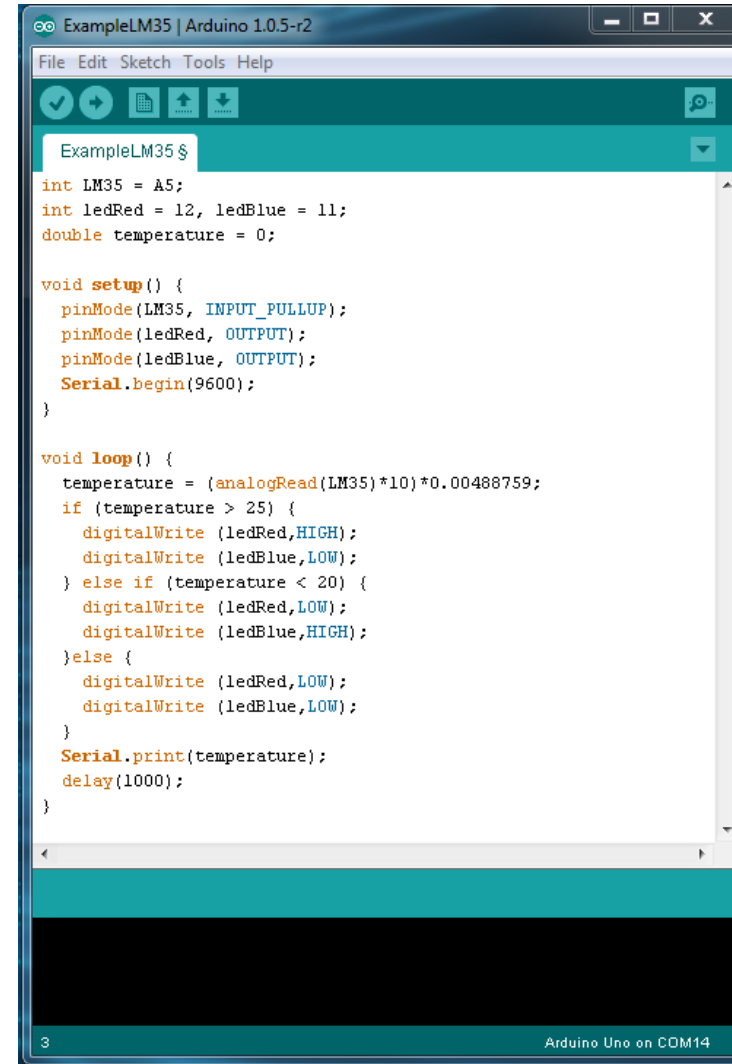
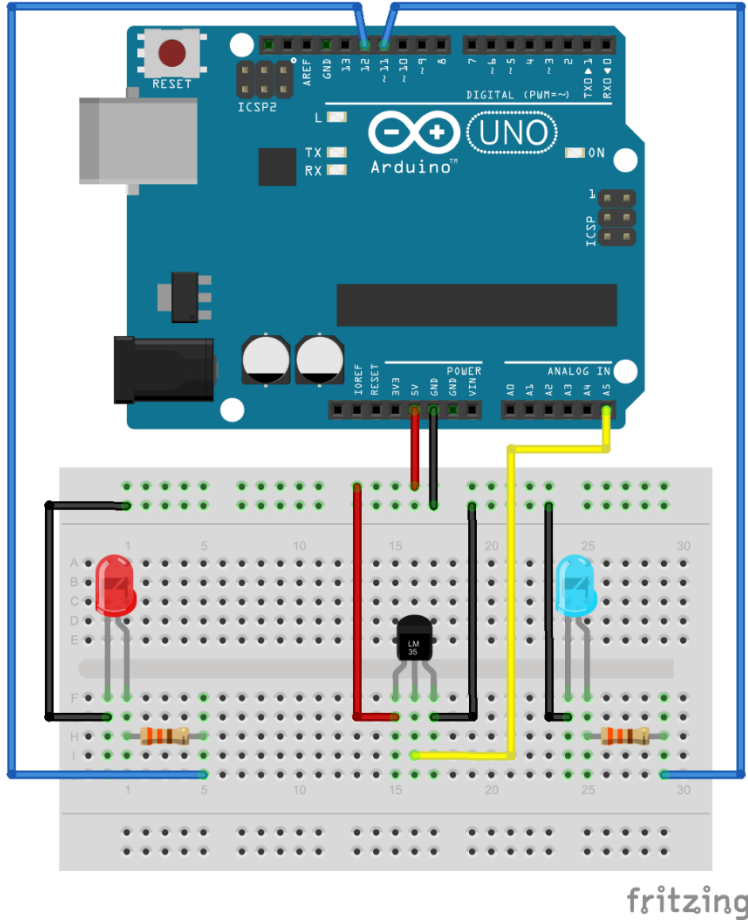
2. ARDUINO: PROJETO SEMÁFORO



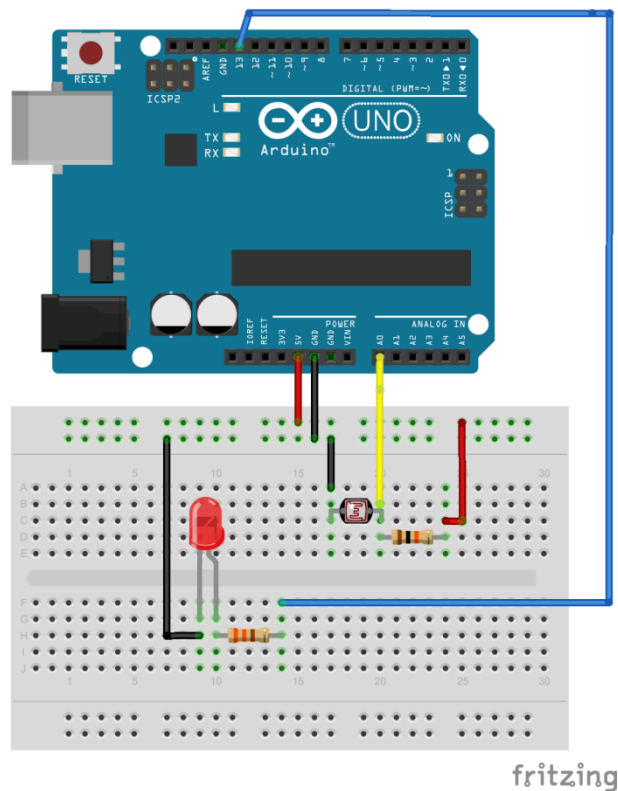
```
int ledVermelho = 11, ledAmarelo = 10, ledVerde = 9;
```

```
void setup () {  
  pinMode (ledVermelho, OUTPUT);  
  pinMode (ledAmarelo, OUTPUT);  
  pinMode (ledVerde, OUTPUT);  
}  
void loop () {  
  digitalWrite (ledVermelho, HIGH);  
  delay (10000);  
  digitalWrite (ledVermelho, LOW);  
  digitalWrite (ledVerde, HIGH);  
  delay (10000);  
  digitalWrite (ledVerde, LOW);  
  digitalWrite (ledAmarelo, HIGH);  
  delay (5000);  
  digitalWrite (ledAmarelo, LOW);  
}
```

2. ARDUINO: PROJETO LM35



2. ARDUINO: PROJETO LDR



```
ExampleLDR | Arduino 1.0.5-r2
File Edit Sketch Tools Help

ExampleLDR
int led = 13;
int LDR = A0;

void setup() {
  pinMode(led, OUTPUT);
  pinMode(LDR, INPUT);
}

void loop() {
  int light = analogRead(LDR);
  if (light > 300) {
    digitalWrite(led, HIGH);
  }
  else {
    digitalWrite(led, LOW);
  }
}

Done Saving.
ExampleLDR.cpp
C:\Users\usuario\AppData\Local\Temp\build5888434680526095542.tmp\
ExampleLDR.cpp.hex
Binary sketch size: 976 bytes (of a 32.256 byte maximum)

17 Arduino Uno on COM14
```

OUTLINE

1. Introdução

2. Usando o Arduino

3. Javino

4. Exemplos

5. Conclusão

Referências Bibliográficas

3. JAVINO

- O Javino é um protocolo que:
 - **Permite a troca de mensagem entre hardware e linguagens de programação de alto nível;**
 - É composto por dupla biblioteca para comunicação serial;
 - Provê detecção de erros através de uma rotina de verificação da recepção de dados.

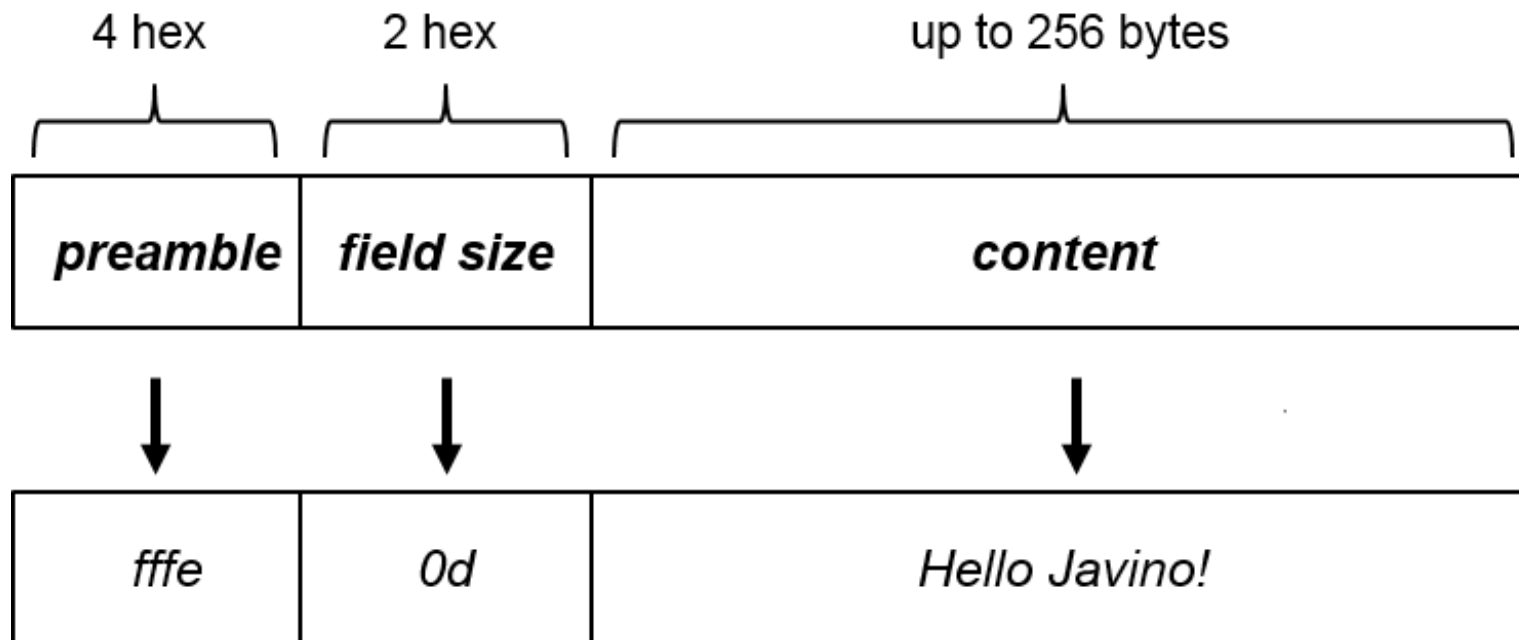
3. JAVINO

- O Javino é um protocolo que:
 - Permite a troca de mensagem entre hardware e linguagens de programação de alto nível;
 - **É composto por dupla biblioteca para comunicação serial;**
 - Provê detecção de erros através de uma rotina de verificação da recepção de dados.

3. JAVINO

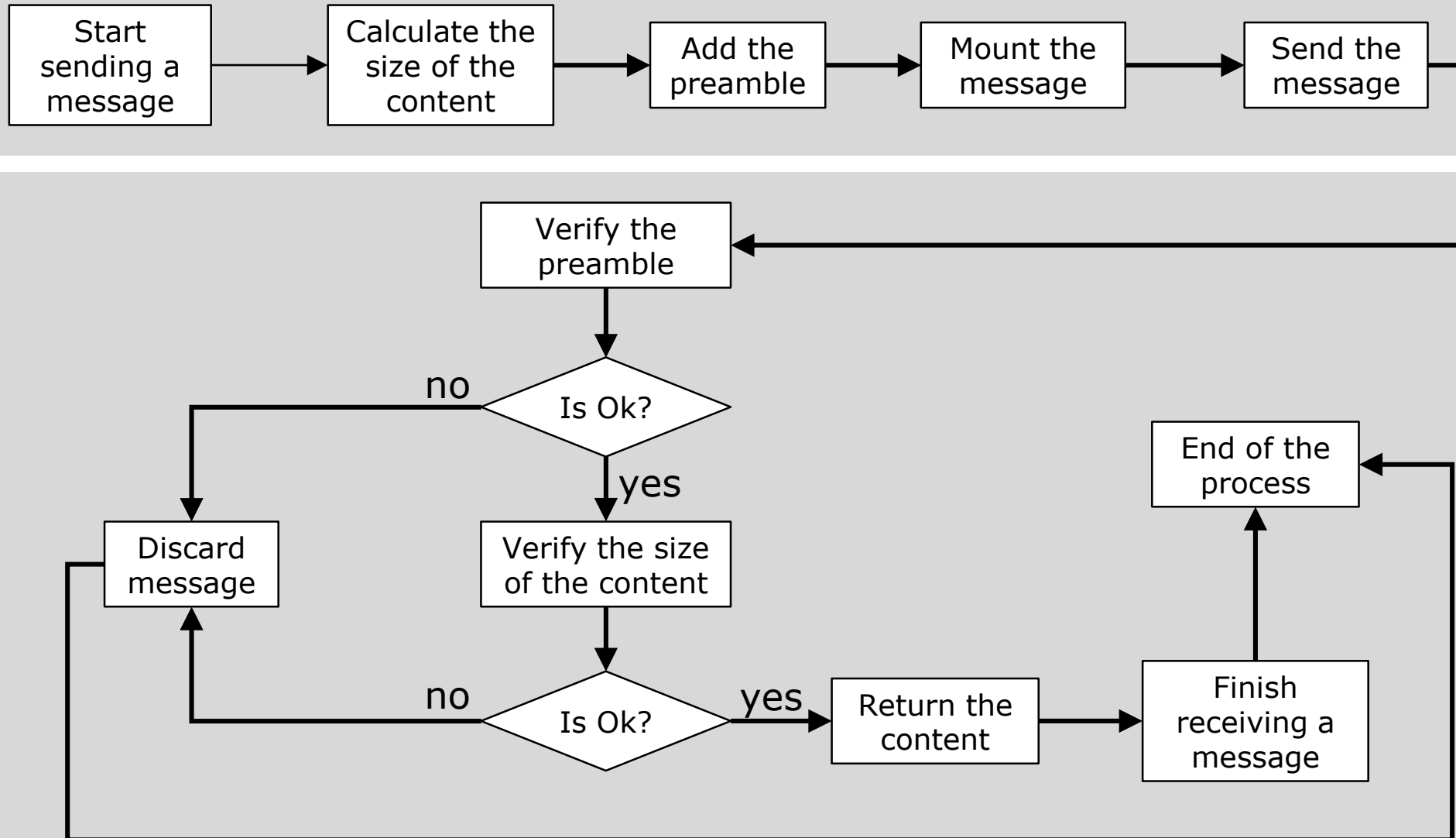
- O Javino é um protocolo que:
 - Permite a troca de mensagem entre hardware e linguagens de programação de alto nível;
 - É composto por dupla biblioteca para comunicação serial;
 - **Provê detecção de erros através de uma rotina de verificação da recepção de dados.**

3. JAVINO: FORMATO DA MENSAGEM



3. JAVINO: FLUXO DA MENSAGEM

SENDER

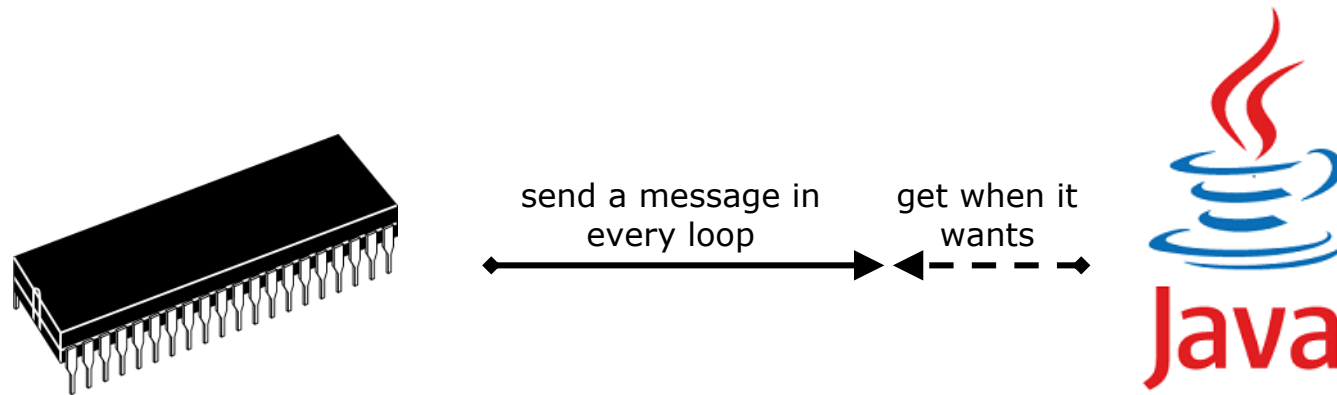


RECEIVER

3. JAVINO: MODOS DE OPERAÇÃO

- **Listen Mode**

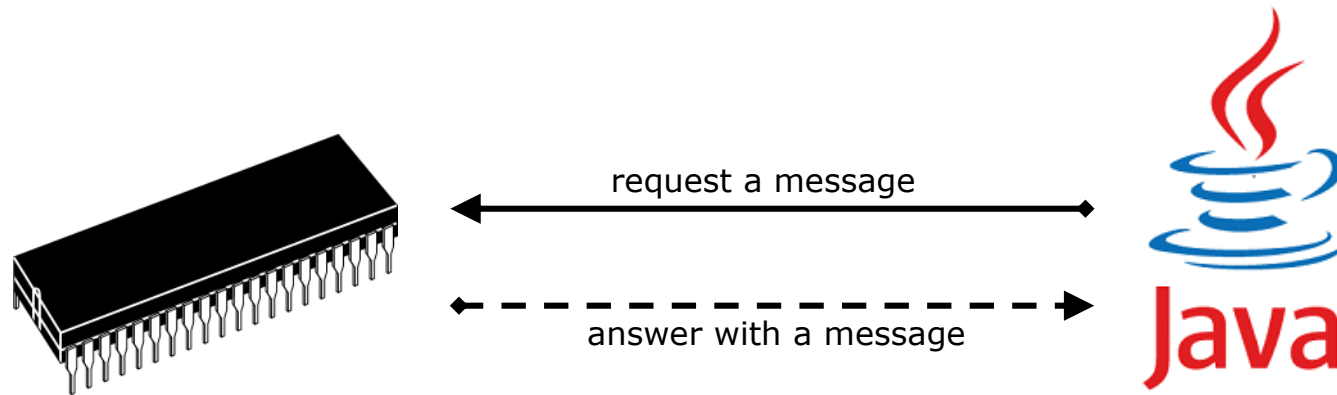
- Mensagens somente do hardware para o software



3. JAVINO: MODOS DE OPERAÇÃO

- **Request Mode**

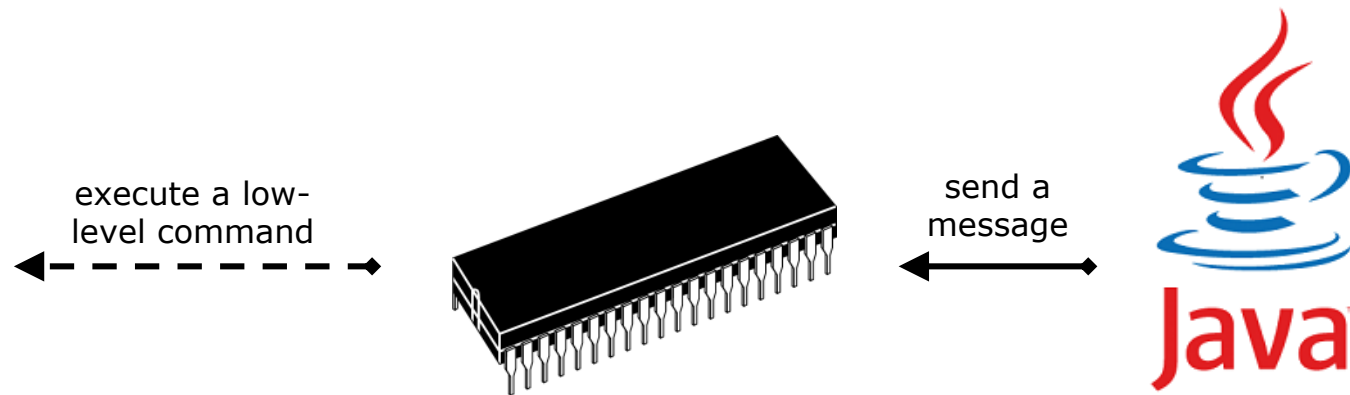
- do software para o hardware;
- o hardware responde com uma mensagem.



3. JAVINO: MODOS DE OPERAÇÃO

- **Send Mode**

- do software para o hardware;
- o hardware executa uma ação.



3. JAVINO: PYTHON E PYSERIAL

- Embora o Python seja conhecido como linguagem de programação orientada a objetos, o Javino utiliza ele e o PySerial para **comunicação de nível inferior** com a porta serial.

3. JAVINO: APLICAÇÃO EM JAVA



Para **iniciar a comunicação em JAVA**, cria-se a instância Javino, que depende do caminho de instalação do Python:

Javino j = **new** Javino

Sem parâmetro (): o Javino considera que o Python está instalado na máquina no endereço Bash padrão.

com parâmetro (pathBash): o Javino considera que o Python está instalado na máquina no endereço Bash determinado pela String **pathBash**.

3. JAVINO: APLICAÇÃO EM JAVA



Além disso, o Javino fornece **recursos** de verificação, leitura e escrita que são chamados nos seguintes métodos em java:

`j.sendCommand(port, msg);`

INDICA A **PORTA DE ENTRADA** DO ARDUINO.

MENSAGEM A SER ENVIADA AO ARDUINO LIGADO À PORTA DETERMINADA.

Descrição

Envia uma mensagem de até **256 caracteres** ao Arduino. No caminho, o Javino forma o **preâmbulo** e a **indicação da quantidade de caracteres de mensagem**. É retornado um **boolean** indicando um feedback do desempenho da comunicação.

3. JAVINO: APLICAÇÃO EM JAVA



Além disso, o Javino fornece **recursos** ao Java que são chamados nos seguintes métodos:

```
j.requestData(port, msg);
```

INDICA A **PORTA DE ENTRADA** DO ARDUINO.

MENSAGEM A SER ENVIADA AO ARDUINO LIGADO À PORTA DETERMINADA.

Descrição

Método que captura uma mensagem vinda do arduino e retorna **true**, caso a mensagem esperada em argumento tenha chegado com sucesso.

3. JAVINO: APLICAÇÃO EM JAVA



Além disso, o Javino fornece **recursos** ao Java que são chamados nos seguintes métodos:

```
j.listenArduino(port);
```

INDICA A PORTA DO ARDUINO
ESCUTADA PELO JAVINO.

Descrição

Retorna **true**, caso haja uma mensagem válida enviada pelo arduino na porta onde o Javino está esperando.

3. JAVINO: APLICAÇÃO EM JAVA



Além disso, o Javino fornece **recursos** ao Java que são chamados nos seguintes métodos:

`j.getData();`

Descrição

Método que pega a mensagem enviada pelo arduino no formato String de até 256 caracteres.

3. JAVINO: APLICAÇÃO EM JAVA



EXEMPLO:

Para efetuar a aplicação Javino para Java, é preciso:

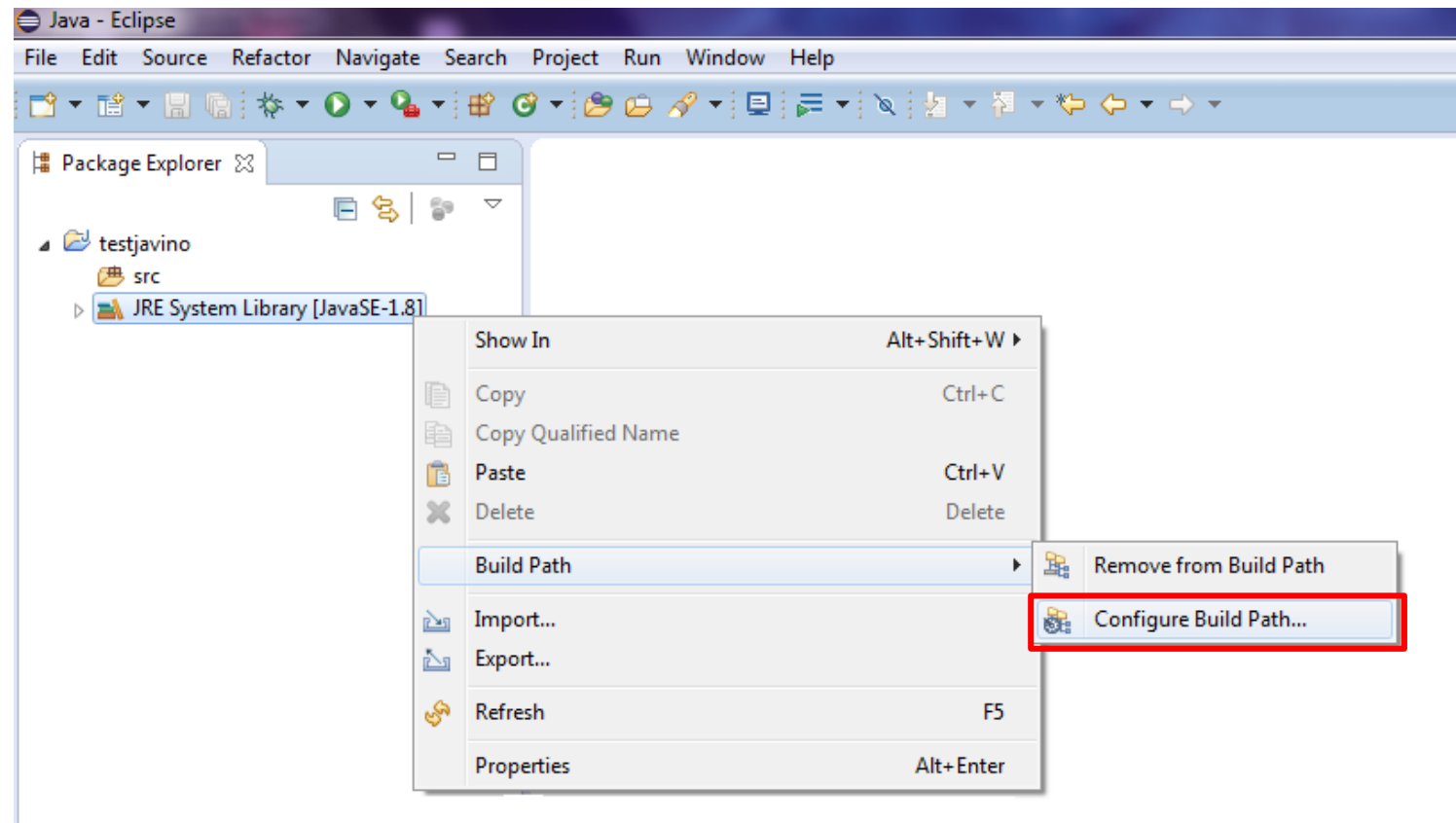
- Obter o Python (<https://www.python.org/>);
- Obter o PySerial (<http://pyserial.sf.net/>);
- Adicionar a biblioteca Javino (<http://javino.sf.net>) para Java em seu projeto;

3. JAVINO: APLICAÇÃO EM JAVA



EXEMPLO:

Em seu projeto, adicione a biblioteca javino para java:

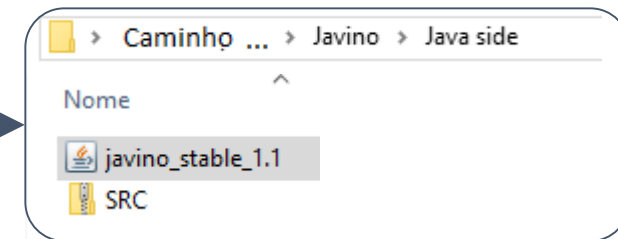
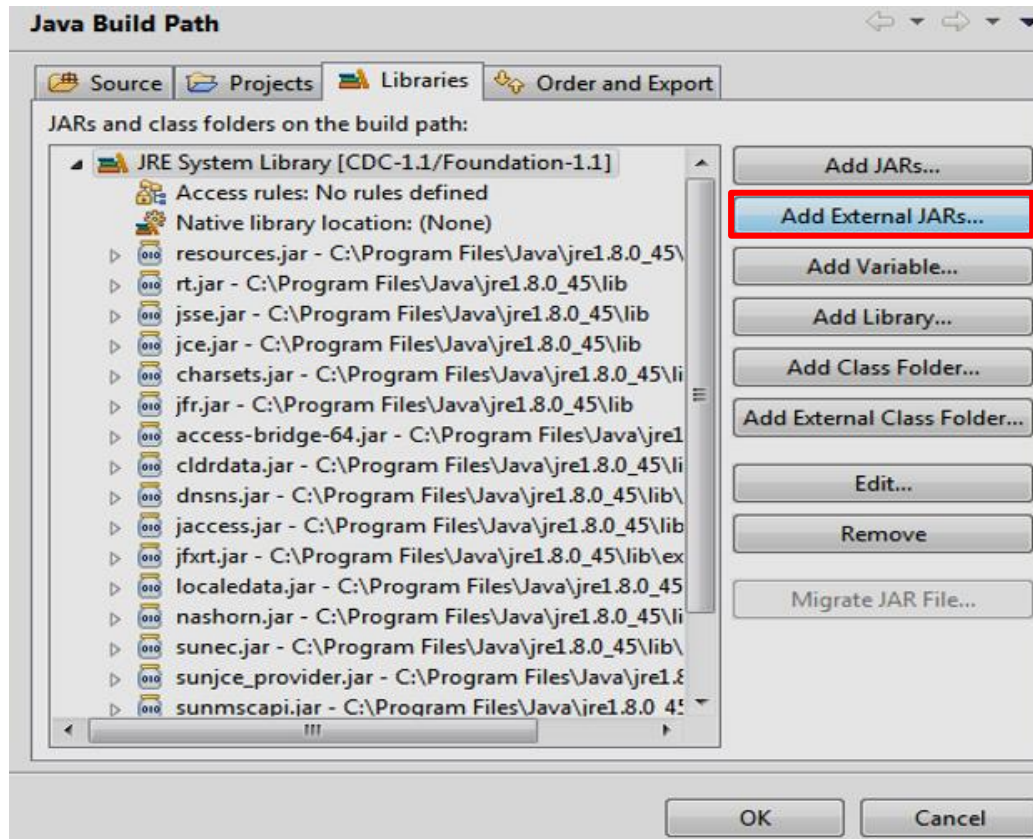


3. JAVINO: APLICAÇÃO EM JAVA



EXEMPLO:

Em seu projeto, adicione a biblioteca javino para java:



3. JAVINO: APLICAÇÃO EM JAVA



EXEMPLO: programando...

```
import br.pro.turing.javino.*;

public class Main {
    public static void main(String args[]) {
        Javino j = new Javino();
        String port = "COM3";

        String ask = "hi";
        if (j.requestData(port, ask)) {
            System.out.println(j.getData());
        }

        ask = "who";
        if (j.requestData(port, ask)) {
            System.out.println(j.getData());
        }

        ask = "status";
        if (j.requestData(port, ask)) {
            System.out.println(j.getData());
        }
    }
}
```

3. JAVINO: APLICAÇÃO EM JAVA



EXEMPLO: programando...

```
import br.pro.turing.javino.*;

public class Main {
    public static void main(String args[]) {
        Javino j = new Javino();
        String port = "COM3";

        String ask = "hi";
        if (j.requestData(port, ask)) {
            System.out.println(j.getData());
        }

        ask = "who";
        if (j.requestData(port, ask)) {
            System.out.println(j.getData());
        }

        ask = "status";
        if (j.requestData(port, ask)) {
            System.out.println(j.getData());
        }
    }
}
```

3. JAVINO: APLICAÇÃO EM JAVA



EXEMPLO: programando...

```
import br.pro.turing.javino.*;

public class Main {
    public static void main(String args[]) {
        Javino j = new Javino();
        String port = "COM3";

        String ask = "hi";
        if (j.requestData(port, ask)) {
            System.out.println(j.getData());
        }

        ask = "who";
        if (j.requestData(port, ask)) {
            System.out.println(j.getData());
        }

        ask = "status";
        if (j.requestData(port, ask)) {
            System.out.println(j.getData());
        }
    }
}
```

3. JAVINO: APLICAÇÃO EM JAVA



EXEMPLO: programando...

```
import br.pro.turing.javino.*;

public class Main {
    public static void main(String args[]) {
        Javino j = new Javino();
        String port = "COM3";

        String ask = "hi";
        if (j.requestData(port, ask)) {
            System.out.println(j.getData());
        }

        ask = "who";
        if (j.requestData(port, ask)) {
            System.out.println(j.getData());
        }

        ask = "status";
        if (j.requestData(port, ask)) {
            System.out.println(j.getData());
        }
    }
}
```

3. JAVINO: APLICAÇÃO EM JAVA



Para **iniciar a comunicação em ARDUINO**:

- inclui-se a **biblioteca** Javino;
- cria-se uma **variável** do tipo Javino para manipulação da comunicação com o Java;
- determina-se a **velocidade da comunicação** pela porta serial;

```
#include <Javino.h>
Javino j;

void setup(){
    Serial.begin(9600);
}
```


3. JAVINO: APLICAÇÃO EM JAVA



Além disso, o Javino fornece **recursos** de verificação, leitura e escrita que são chamados nas seguintes funções Arduino:

`j.sendmsg(msg);` → Envia uma **mensagem de até 256 caracteres** ao Java.

3. JAVINO: APLICAÇÃO EM JAVA



Além disso, o Javino fornece **recursos** de verificação, leitura e escrita que são chamados nas seguintes funções Arduino:

`j.availablemsg();`



Retorna um boolean indicando se há mensagem válida disponível no buffer de entrada.

3. JAVINO: APLICAÇÃO EM ARDUINO



Para efetuar a aplicação Javino pelo Arduino, é preciso:

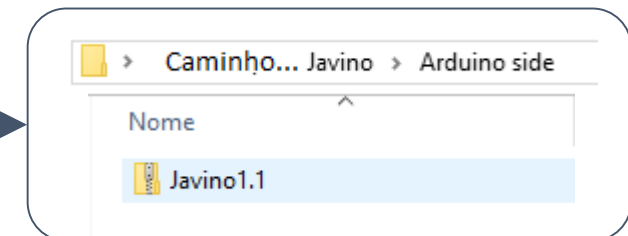
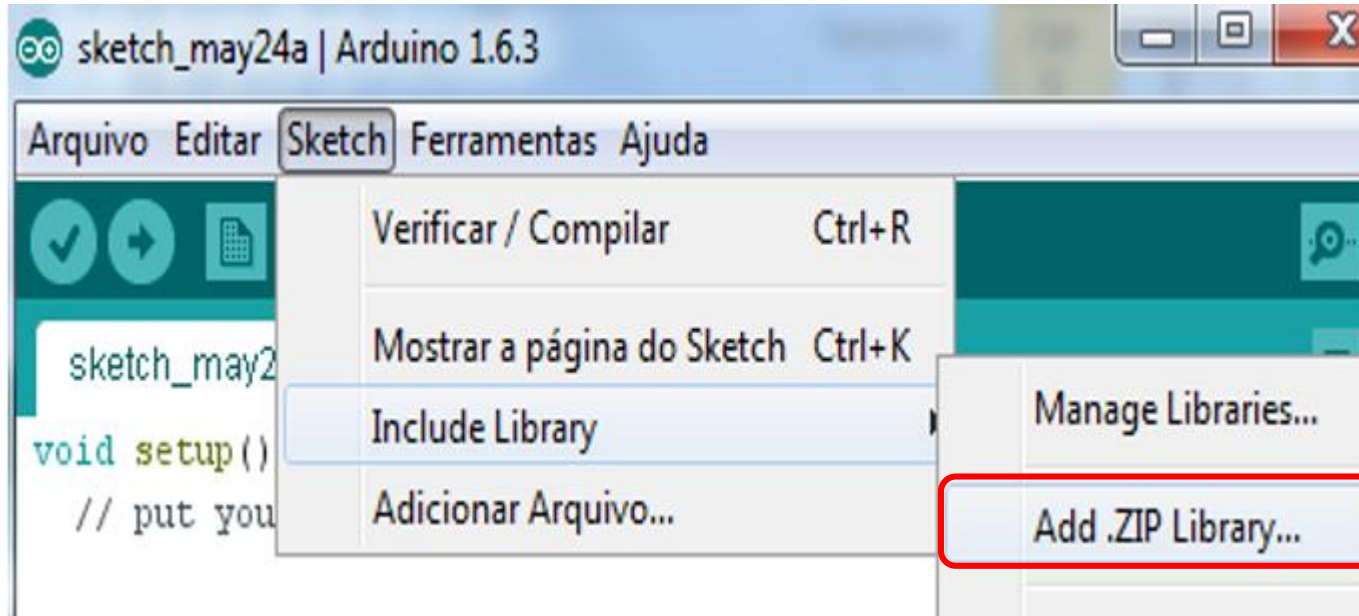
- Adicionar a biblioteca Javino (<http://javino.sf.net>) para Java em seu projeto;
- Em seu Hardware, durante a execução do código, de um capacitor entre o pino reset e GND, que vem contido no Kit Javino;

3. JAVINO: APLICAÇÃO EM ARDUINO

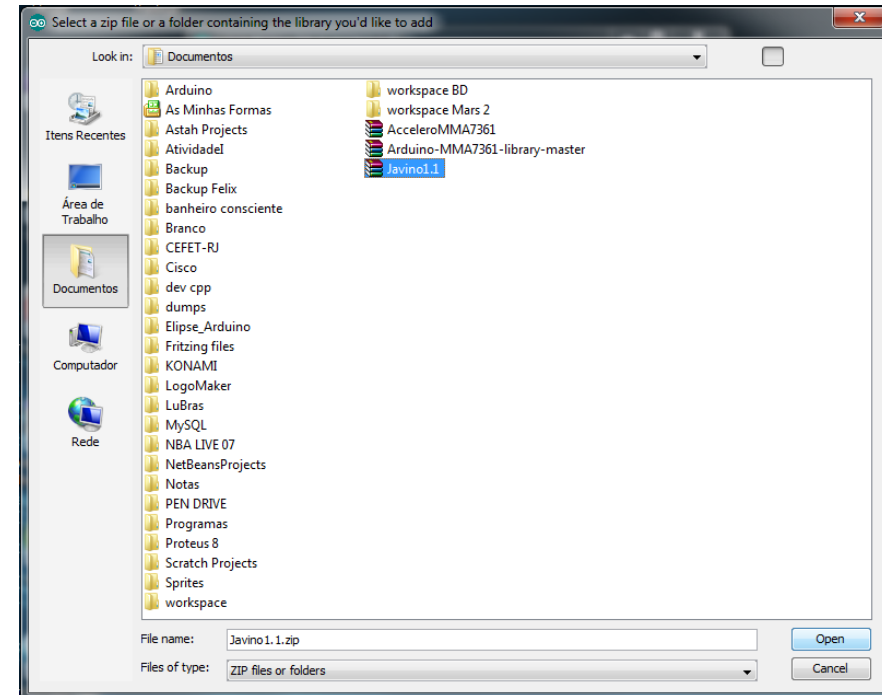
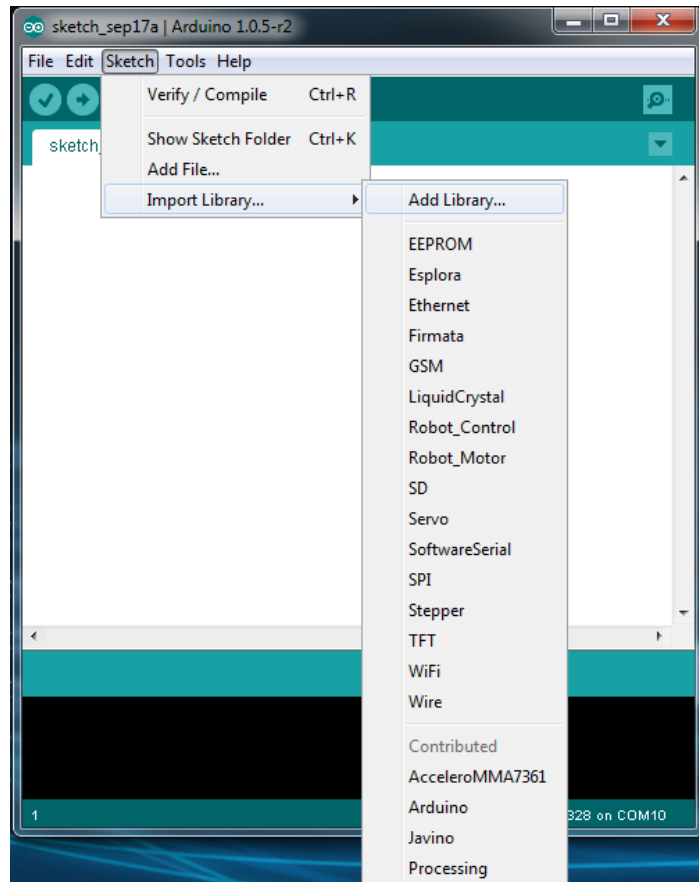


EXEMPLO:

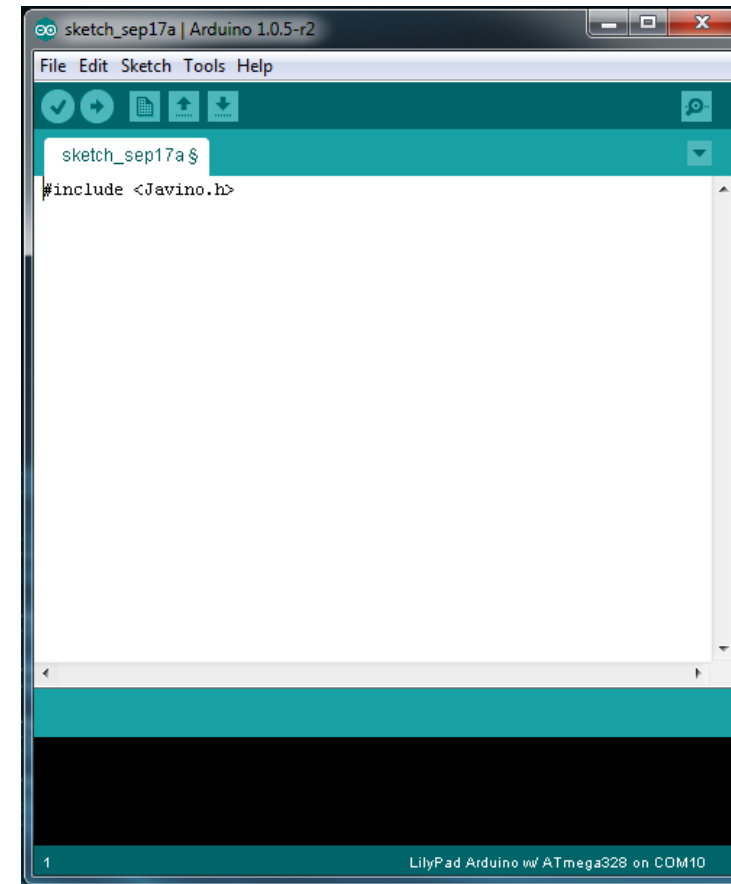
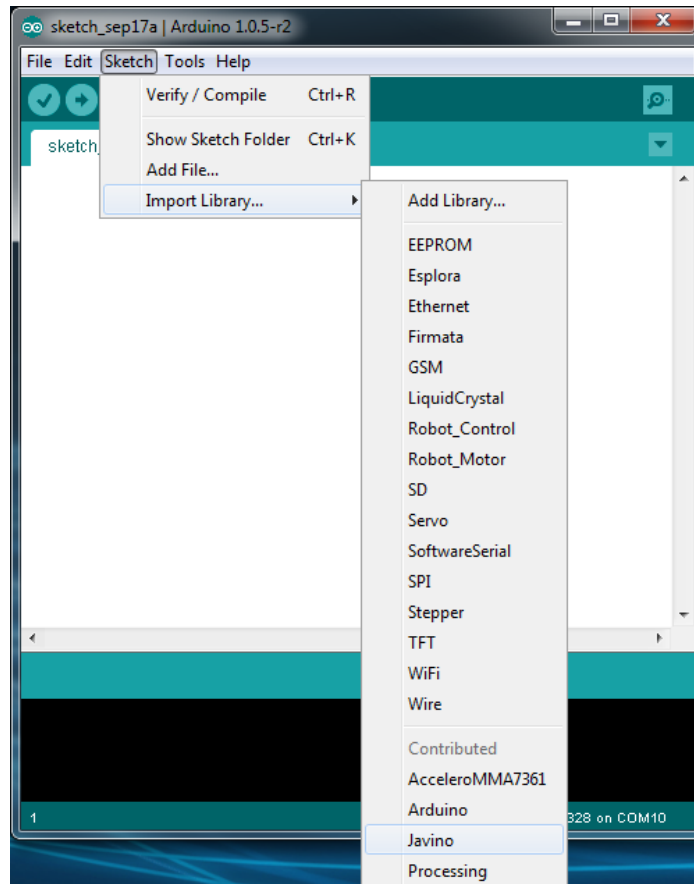
Em seu projeto, adicione a biblioteca Javino para arduino:



3. JAVINO: IMPORTANDO O JAVINO NO ARDUINO



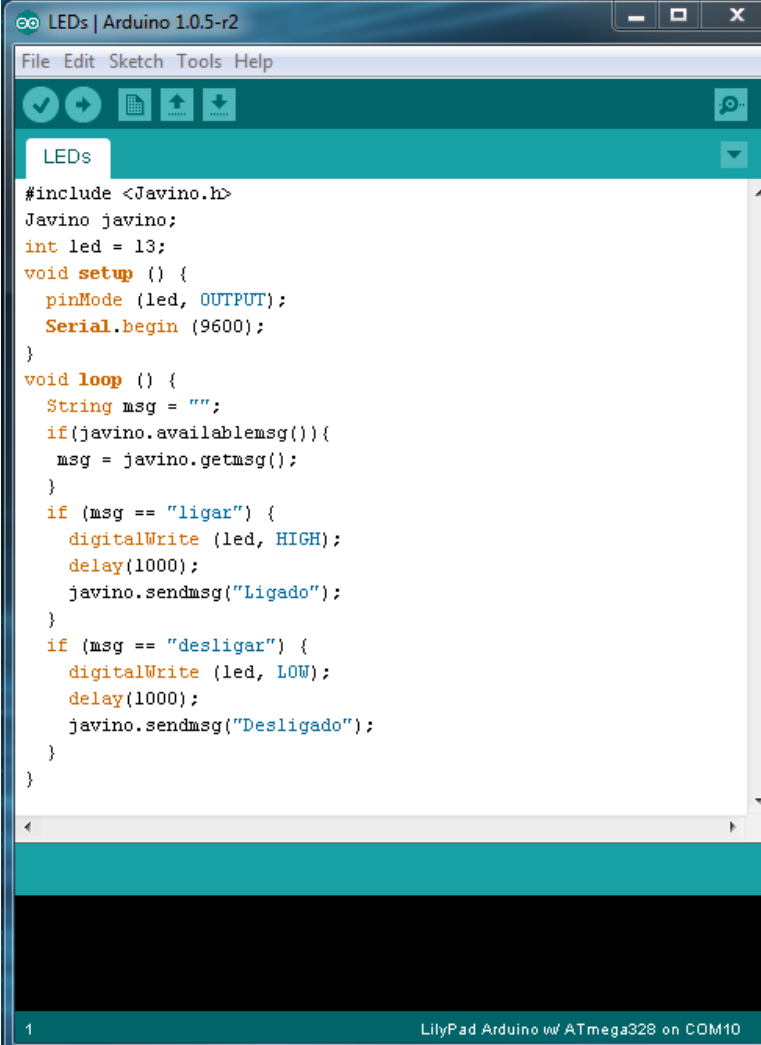
3. JAVINO: IMPORTANDO O JAVINO NO ARDUINO



3. JAVINO: REQUEST MODE NO ARDUINO + JAVA

Acionamento de Led via Javino

Neste exemplo, o arduino **receberá** mensagens externas e **executará** uma determinada tarefa e **retornará** com uma resposta ao comandante.



```
#include <Javino.h>
Javino javino;
int led = 13;
void setup () {
  pinMode (led, OUTPUT);
  Serial.begin (9600);
}
void loop () {
  String msg = "";
  if(javino.availablemsg()){
    msg = javino.getmsg();
  }
  if (msg == "ligar") {
    digitalWrite (led, HIGH);
    delay(1000);
    javino.sendmsg("Ligado");
  }
  if (msg == "desligar") {
    digitalWrite (led, LOW);
    delay(1000);
    javino.sendmsg("Desligado");
  }
}
```

3. JAVINO: REQUEST MODE NO ARDUINO + JAVA

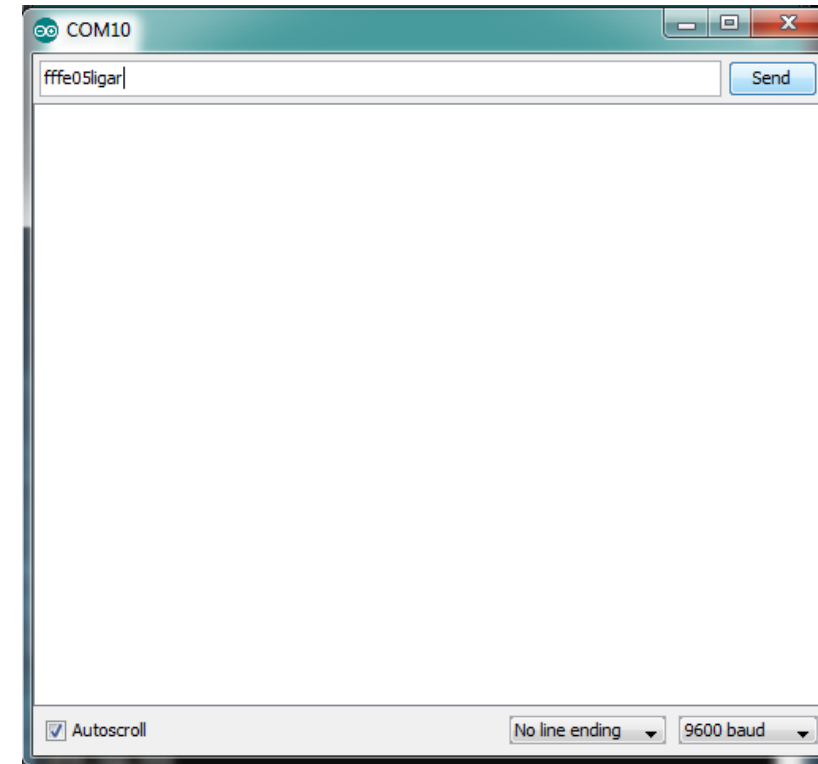
Enviando o comando de **ligar** o Led.

fffe 05 ligar

Pré-âmbulo para
verificação de
recebimento da
mensagem

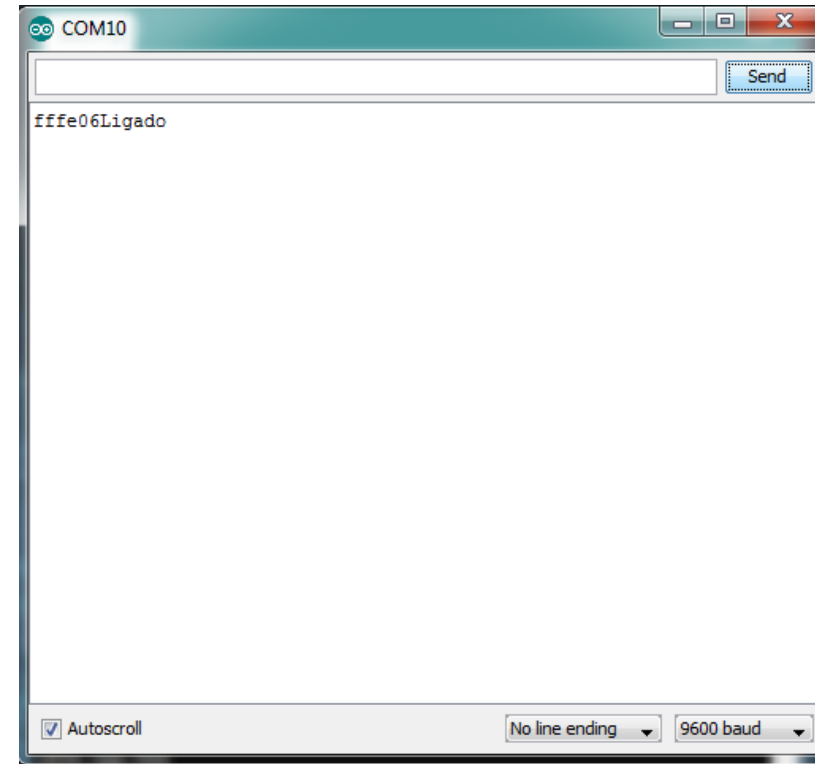
Tamanho do
conteúdo a ser
enviado em
hexadecimal

Conteúdo a ser
enviado



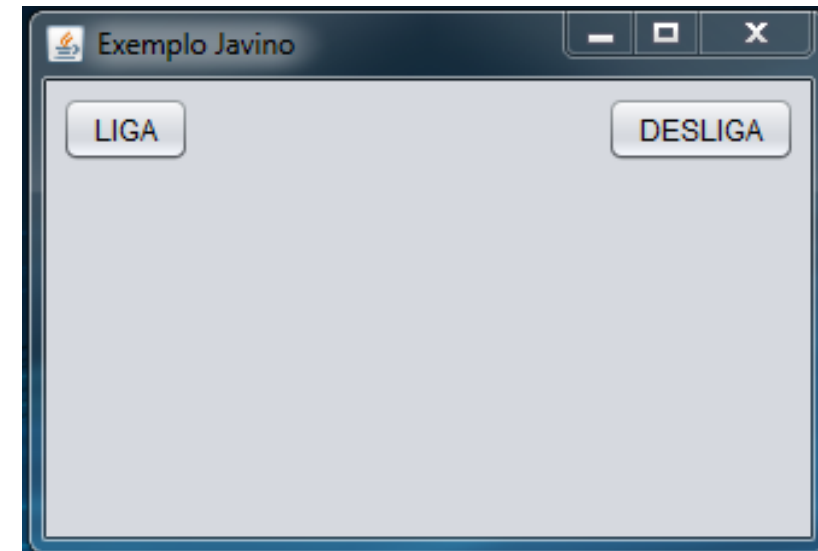
3. JAVINO: REQUEST MODE NO ARDUINO + JAVA

Resultado do comando de **ligar**
o Led.



3. JAVINO: REQUEST MODE NO ARDUINO + JAVA

Comandando através de uma **interface** gráfica em **Java**.

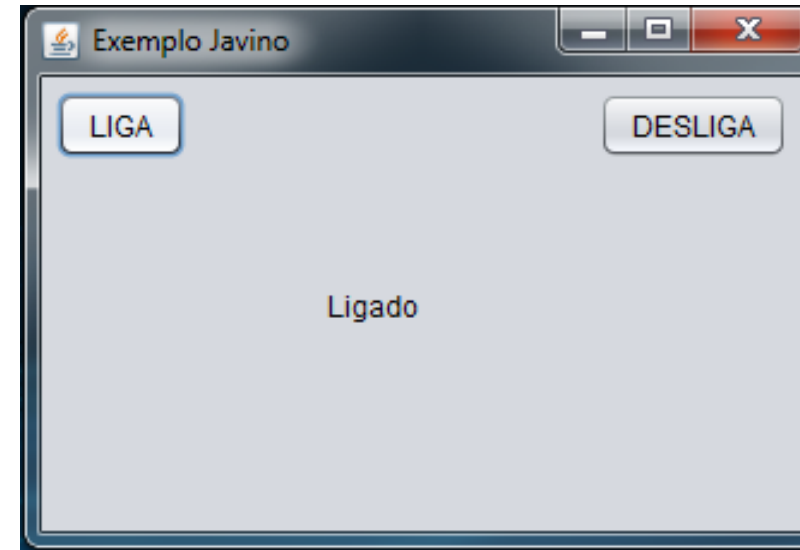


3. JAVINO: REQUEST MODE NO ARDUINO + JAVA

```
1
2 package led;
3 import br.pro.turing.javino.Javino;
4
5 public class Interface extends javax.swing.JFrame {
6
7     private Javino jBridge = new Javino("C:\\Python27");
8     public Interface() {
9         initComponents();
10    }
11
12
13    @SuppressWarnings("unchecked")
14    Generated Code
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29 private void btOnActionPerformed(java.awt.event.ActionEvent evt) {
30     this.jBridge.sendCommand("COM10" , "ligar");
31     if(jBridge.listenArduino("COM10")){
32         lblStatus.setText(jBridge.getData());
33     }
34 }
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86 private void btOffActionPerformed(java.awt.event.ActionEvent evt) {
87     this.jBridge.sendCommand("COM10" , "desligar");
88     if(jBridge.listenArduino("COM10")){
89         lblStatus.setText(jBridge.getData());
90     }
91 }
```

3. JAVINO: REQUEST MODE NO ARDUINO + JAVA

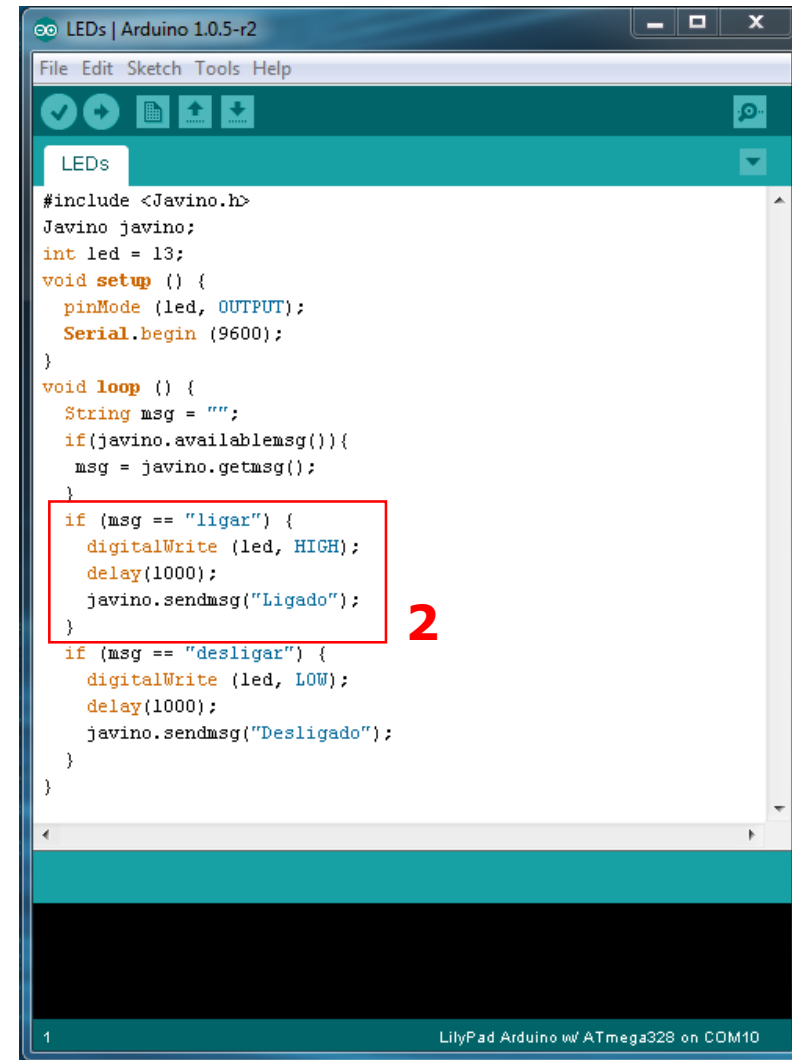
Resultado do botão **ligar**.



3. JAVINO: SEND MODE NO ARDUINO + JAVA

```
1 package led;
2 import br.pro.turing.javino.Javino;
3
4
5 public class Interface extends javax.swing.JFrame {
6
7     private Javino jBridge = new Javino("C:\\Python27");
8     public Interface() {
9         initComponents();
10    }
11
12    @SuppressWarnings("unchecked")
13    Generated Code
14
15    private void btOnActionPerformed(java.awt.event.ActionEvent evt) {
16        this.jBridge.sendCommand("COM10", "ligar");
17        if (jBridge.listenArduino("COM10")) {
18            lblStatus.setText(jBridge.getData());
19        }
20    }
21
22    private void btOffActionPerformed(java.awt.event.ActionEvent evt) {
23        this.jBridge.sendCommand("COM10", "desligar");
24        if (jBridge.listenArduino("COM10")) {
25            lblStatus.setText(jBridge.getData());
26        }
27    }
28 }
```

1



```
LEDs | Arduino 1.0.5-r2
File Edit Sketch Tools Help

#include <Javino.h>
Javino javino;
int led = 13;

void setup () {
    pinMode (led, OUTPUT);
    Serial.begin (9600);
}

void loop () {
    String msg = "";
    if (javino.availablemsg()) {
        msg = javino.getmsg();
    }

    if (msg == "ligar") {
        digitalWrite (led, HIGH);
        delay(1000);
        javino.sendmsg("Ligado");
    }

    if (msg == "desligar") {
        digitalWrite (led, LOW);
        delay(1000);
        javino.sendmsg("Desligado");
    }
}

1 LilyPad Arduino w/ ATmega328 on COM10
```

2

3. JAVINO: LISTEN MODE NO ARDUINO + JAVA

```
1
2 package led;
3 import br.pro.turing.javino.Javino;
4
5 public class Interface extends javax.swing.JFrame {
6
7     private Javino jBridge = new Javino("C:\\Python27");
8     public Interface() {
9         initComponents();
10    }
11
12
13    @SuppressWarnings("unchecked")
14    Generated Code
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29 private void btOnActionPerformed(java.awt.event.ActionEvent evt) {
30     this.jBridge.sendCommand("COM10" , "ligar");
31     if(jBridge.listenArduino("COM10")){
32         lblStatus.setText(jBridge.getData());
33     }
34 }
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86 private void btOffActionPerformed(java.awt.event.ActionEvent evt) {
87     this.jBridge.sendCommand("COM10" , "desligar");
88     if(jBridge.listenArduino("COM10")){
89         lblStatus.setText(jBridge.getData());
90     }
91 }
```

OUTLINE

1. Introdução

2. Usando o Arduino

3. Javino

4. Exemplos

5. Conclusão

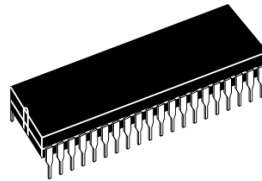
Referências Bibliográficas

4. EXEMPLOS

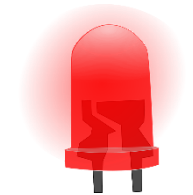
- O Javino precisa ser programado em modo send e request na controladora
- Para cada led um procedimento de ativação deve ser programado em resposta a um estímulo (modo send):



lightOn

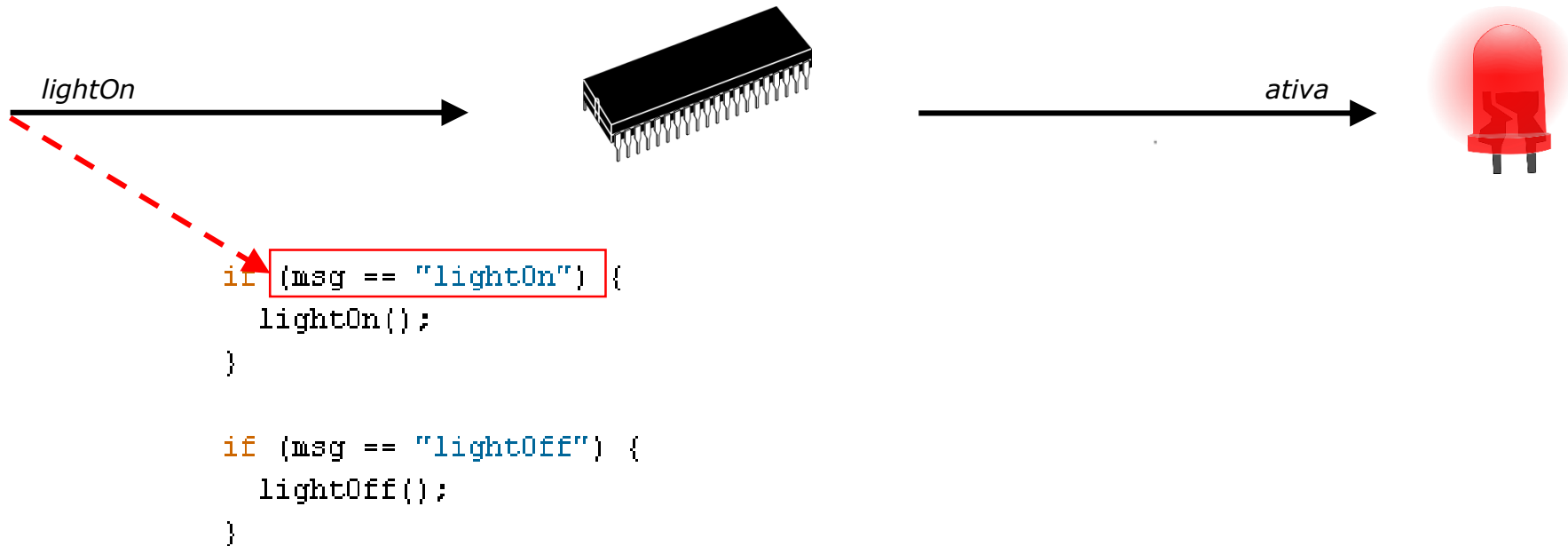


ativa



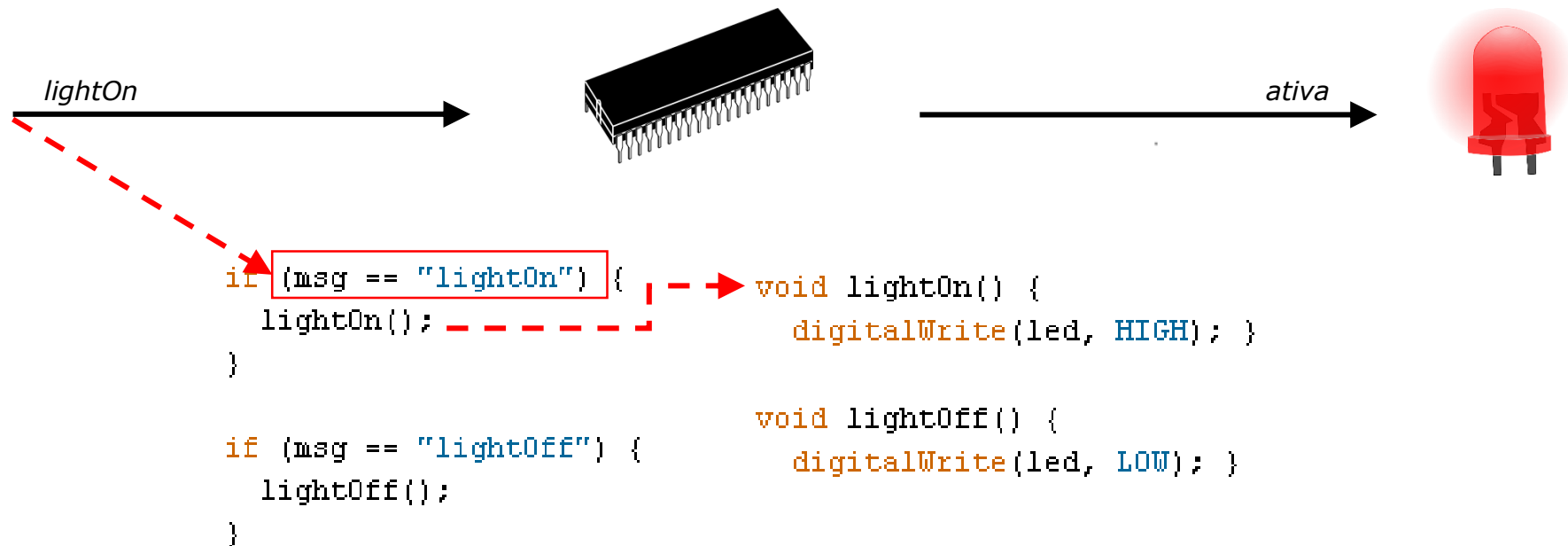
4. EXEMPLOS

- O Javino precisa ser programado em modo send e request na controladora
- Para cada led um procedimento de ativação deve ser programado em resposta a um estímulo (modo send):



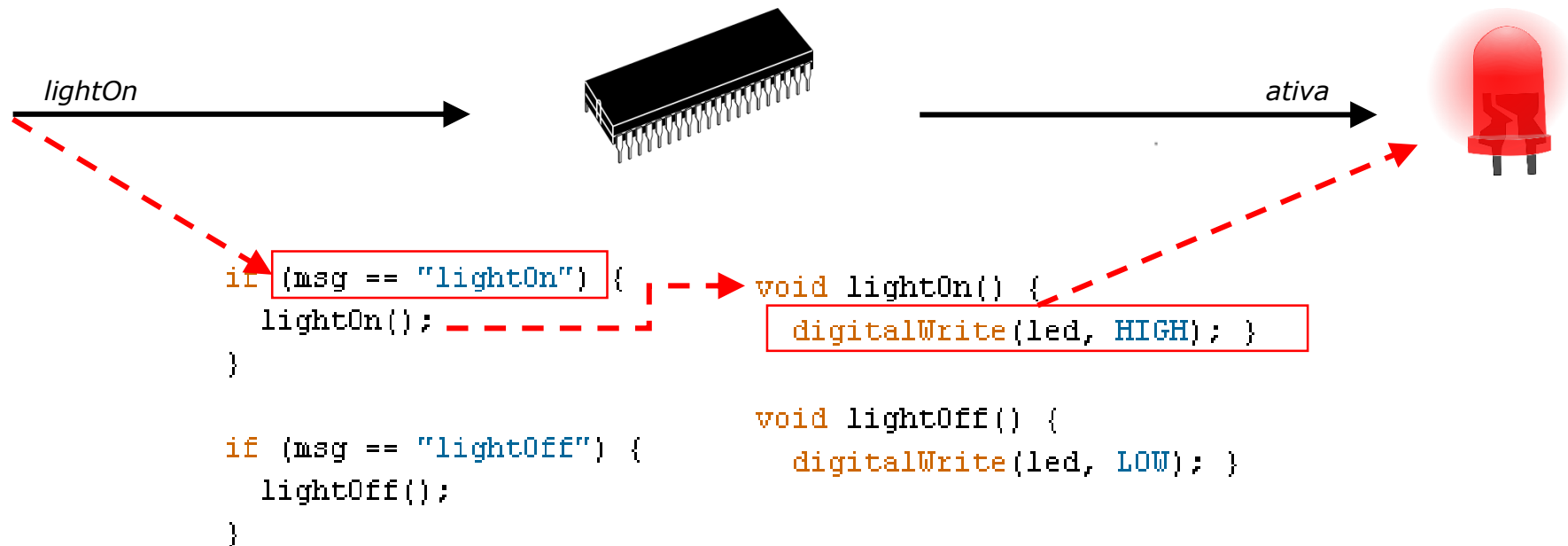
4. EXEMPLOS

- O Javino precisa ser programado em modo send e request na controladora
- Para cada led um procedimento de ativação deve ser programado em resposta a um estímulo (modo send):



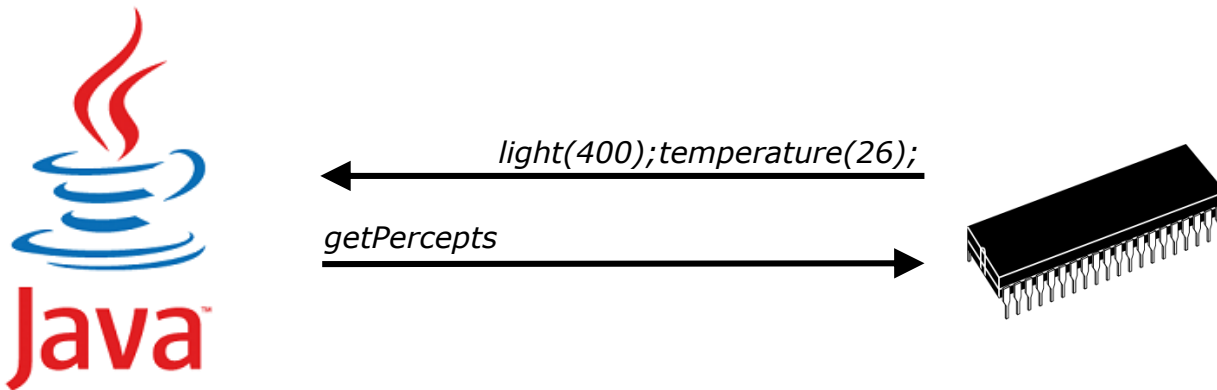
4. EXEMPLOS

- O Javino precisa ser programado em modo send e request na controladora
- Para cada led um procedimento de ativação deve ser programado em resposta a um estímulo (modo send):



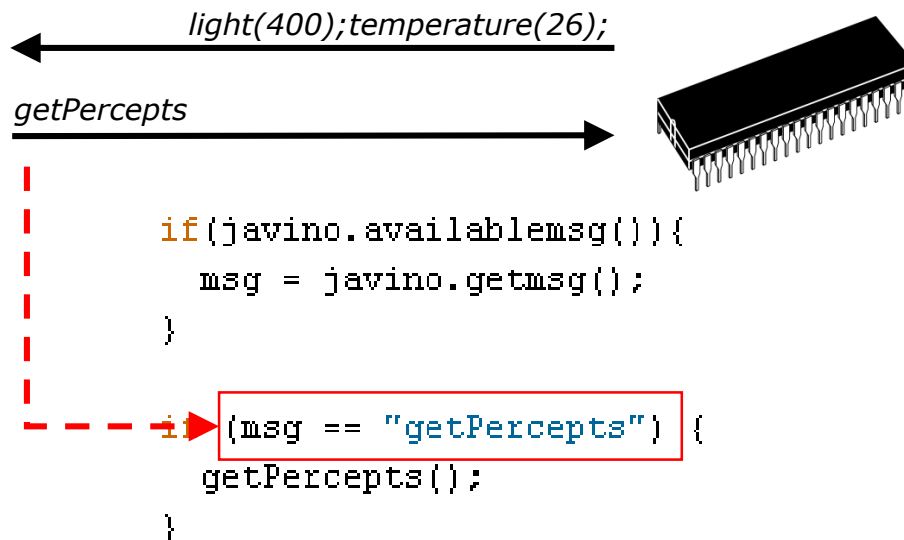
4. EXEMPLOS

- Para isso, um procedimento de envio das percepções deve ser programado em resposta a um estímulo ***getPercepts*** (modo request):



4. EXEMPLOS

- Para isso, um procedimento de envio das percepções deve ser programado em resposta a um estímulo ***getPercepts*** (modo request):

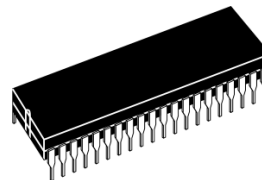


4. EXEMPLOS

- Para isso, um procedimento de envio das percepções deve ser programado em resposta a um estímulo ***getPercepts*** (modo request):



← *light(400);temperature(26);*
getPercepts →



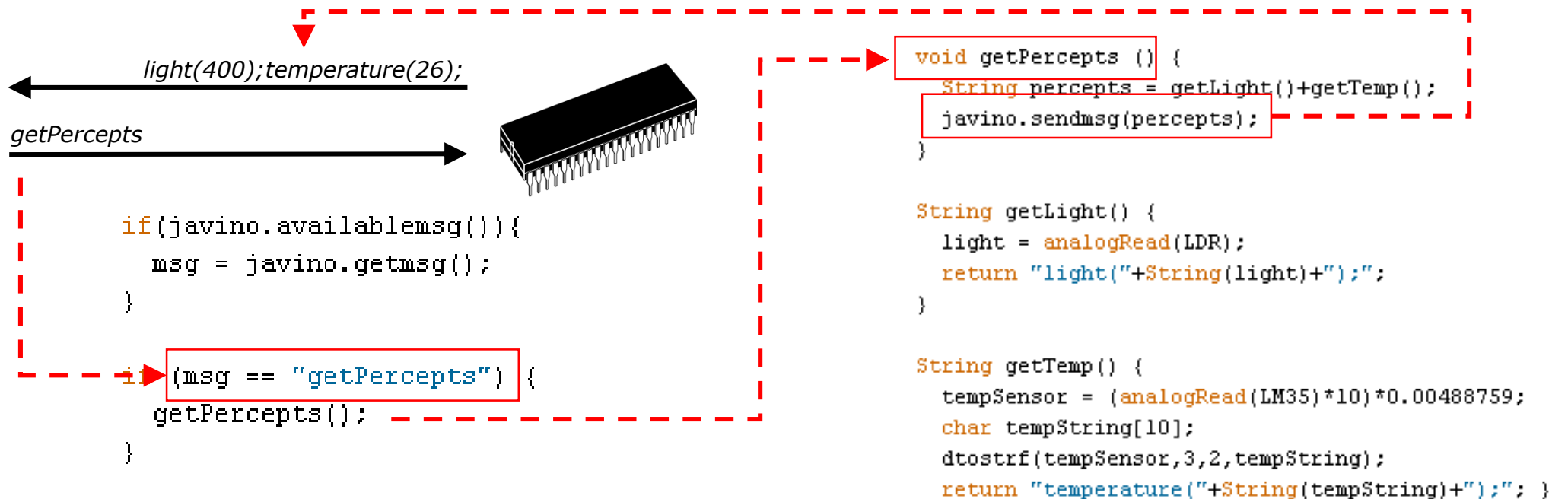
```
if(javino.availablemsg()){  
    msg = javino.getmsg();  
}
```

```
if(msg == "getPercepts") {  
    getPercepts();  
}
```

```
void getPercepts () {  
    String percepts = getLight()+getTemp();  
    javino.sendmsg(percepts);  
}  
  
String getLight() {  
    light = analogRead(LDR);  
    return "light("+String(light)+");";  
}  
  
String getTemp() {  
    tempSensor = (analogRead(LM35)*10)*0.00488759;  
    char tempString[10];  
    dtostrf(tempSensor,3,2,tempString);  
    return "temperature("+String(tempString)+");"; }  
}
```

4. EXEMPLOS

- Para isso, um procedimento de envio das percepções deve ser programado em resposta a um estímulo ***getPercepts*** (modo request):

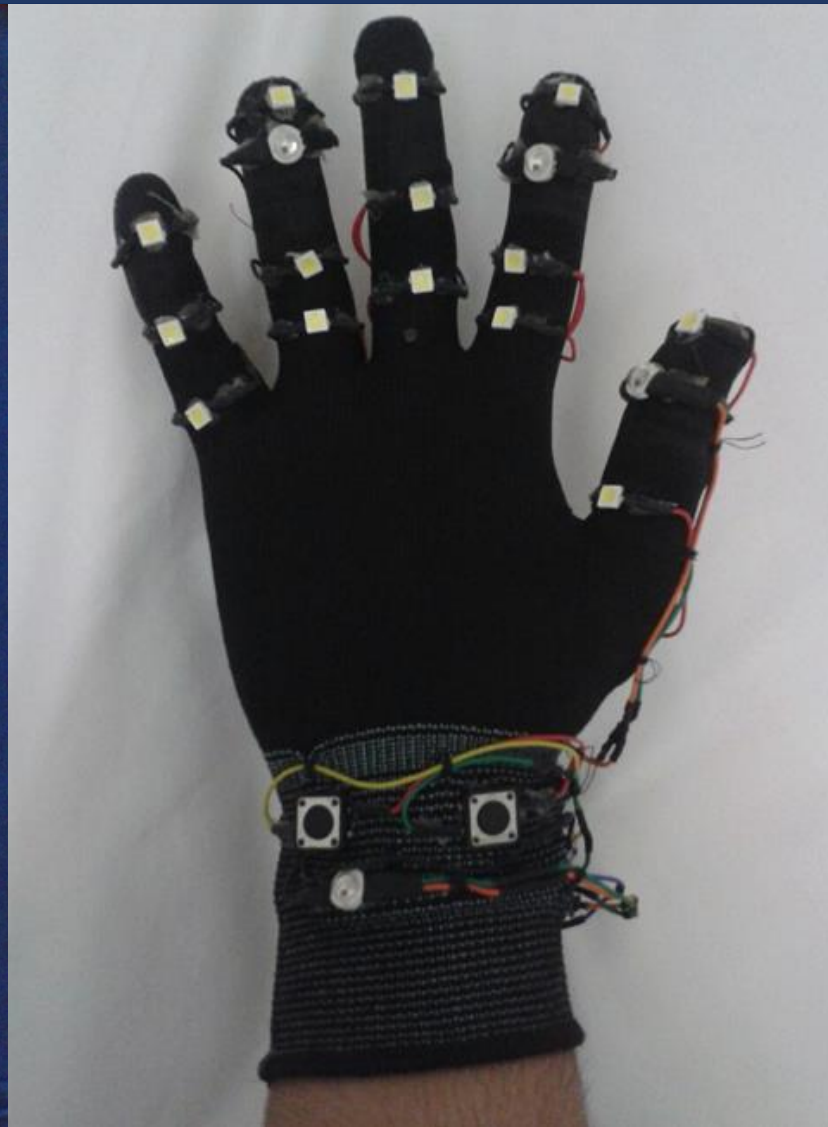
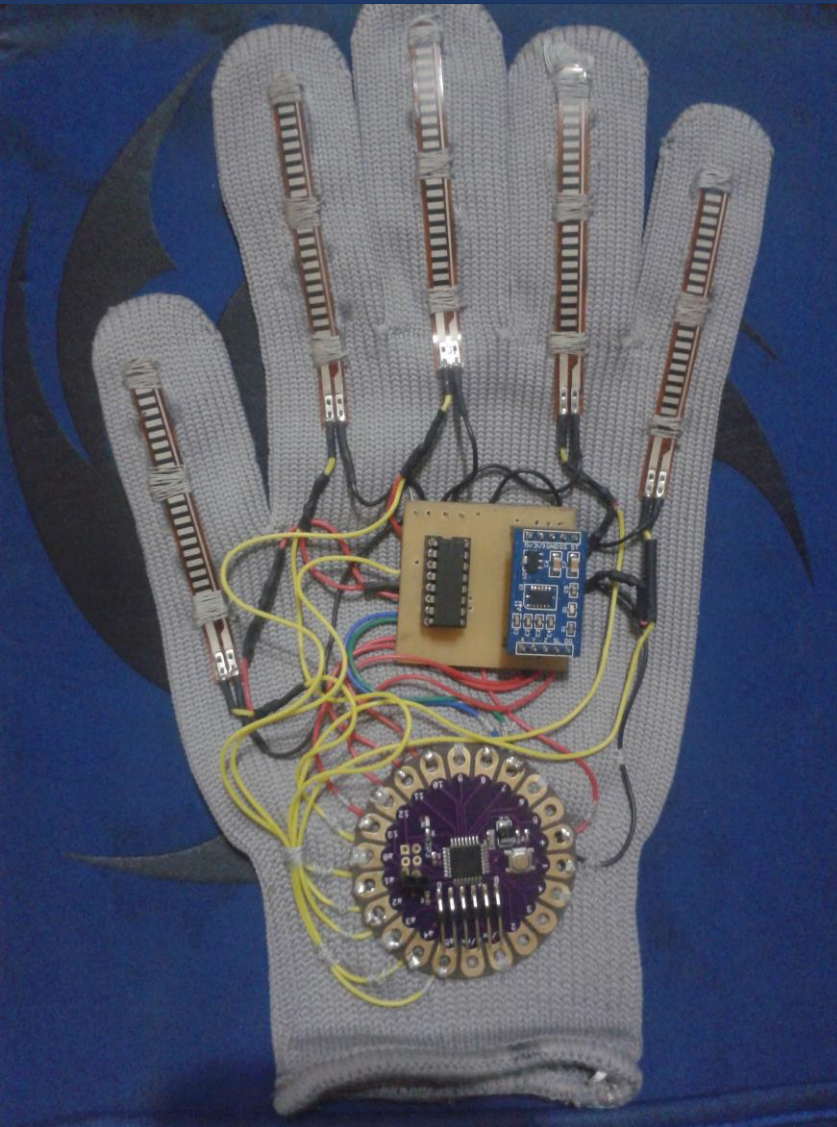


4. EXEMPLOS

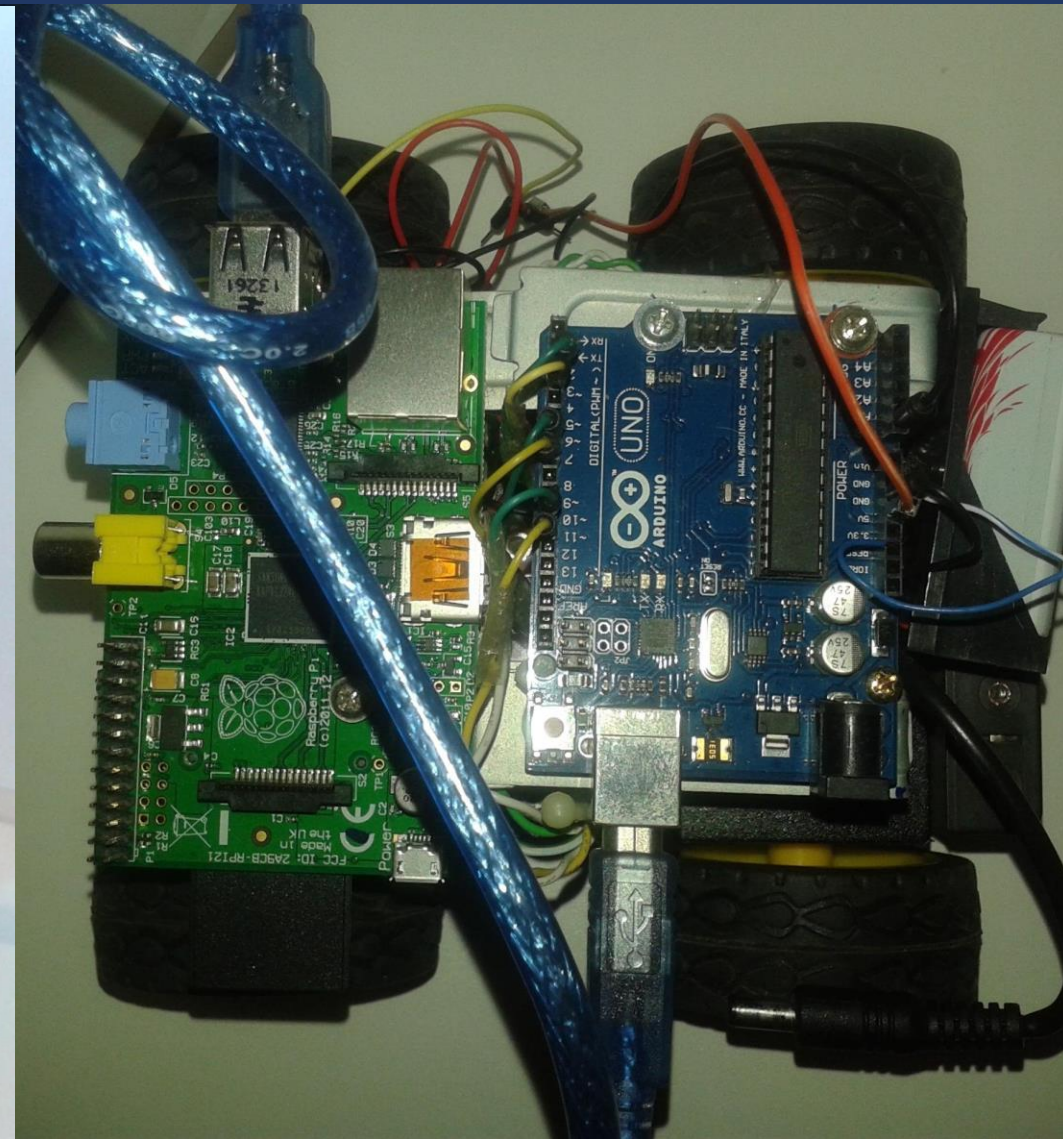
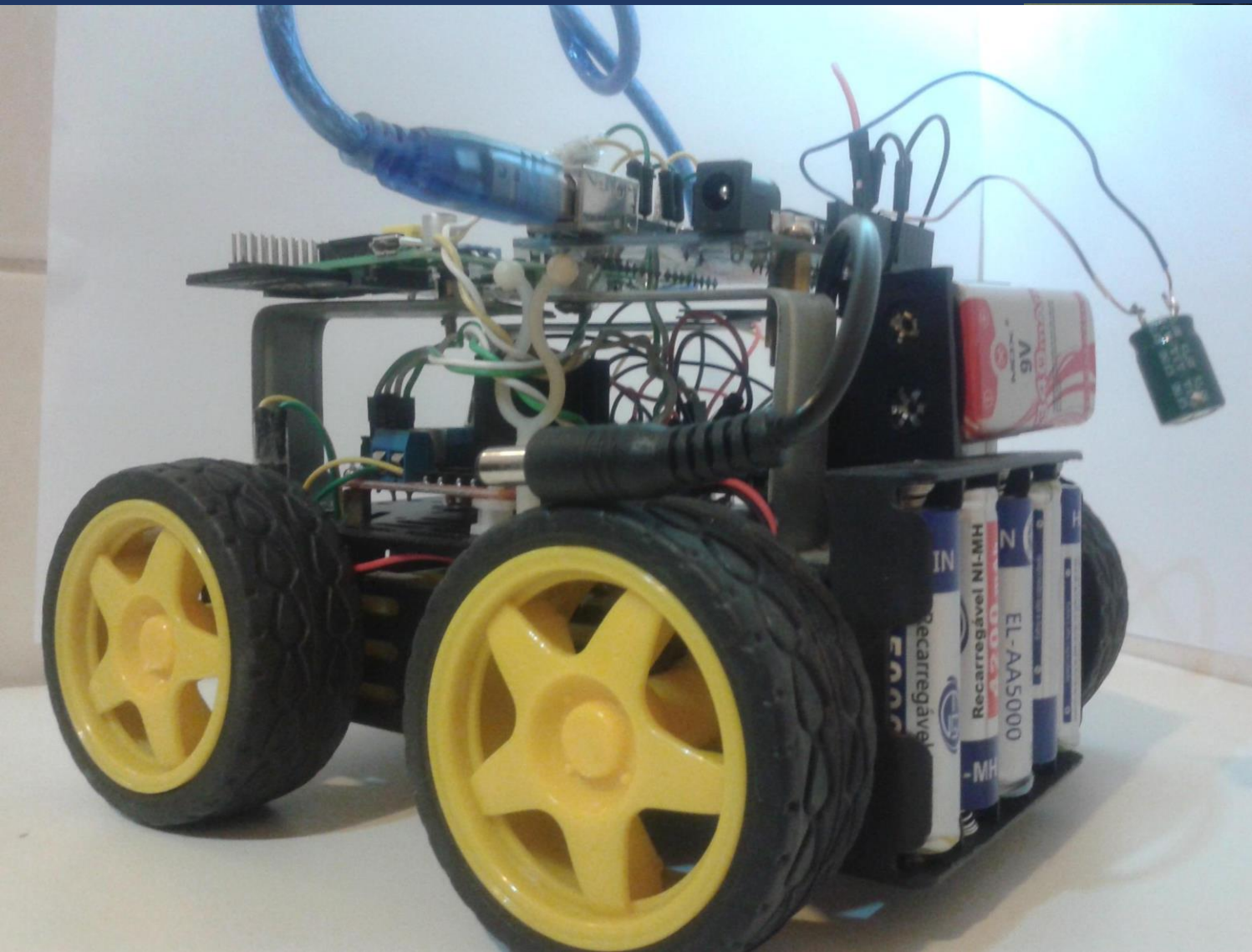
- **Atenção para:**

- i. Remover a mensagem recebida pelo Javino a cada loop da controladora;
- ii. Importar a biblioteca do Javino do lado do Arduino;
- iii. Após a programação, colocar o capacitor no Arduino (+ no RESET e – no GND).

4. EXEMPLOS: LUBRAS



4. EXEMPLOS: INTELIGÊNCIA ARTIFICIAL - AGENTES



OUTLINE

1. Introdução

2. Usando o Arduino

3. Javino

4. Exemplos

5. Conclusão

Referências Bibliográficas

5. CONCLUSÃO

Neste mini-curso foi mostrado como se controlar um Arduino através da utilização do Java.

OUTLINE

1. Introdução

2. Usando o Arduino

3. Javino

4. Exemplos

5. Conclusão

Referências Bibliográficas

REFERÊNCIAS BIBLIOGRÁFICA

- [Bordini et al. 2007] Bordini, R.H., Hubner, J.F., Wooldridge, M. Programming Multi-Agent Systems in AgentSpeak Using Jason. John Wiley & Sons Ltd., 2007.
- [Bratman, 1987] Bratman, M. Intentions, Plans, and Practical Reason. Harvard University Press, 1987.
- [Guinelli et al., 2016] Guinelli, J. V. ; Junger, D. S. ; Pantoja, C. E. . An Analysis of Javino Middleware for Robotic Platforms Using Jason and JADE Frameworks. In: Workshop-Escola de Sistemas de Agentes, Seus Ambientes e Aplicações, Maceió. Anais do X Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações, 2016.
- [Huber, 1999] Huber MJ. Jam: a bdi-theoretic mobile agent architecture. In Proceedings of the third annual conference on Autonomous Agents, AGENTS '99, pags. 236-243, New York, 1999
- [Lazarin and Pantoja, 2015] Lazarin, N.M., Pantoja, C.E. : A robotic-Agent Platform For Embedding Software Agents Using Raspberry Pi and Arduino Boards. In: 9th Software Agents, Environments and Applications School, 2015
- [Pantoja et al., 2016] Pantoja, C. E.; Stabile Jr, M. F. ; Lazarin, N. M. ; Sichman, J. S. ARGO: A Customized Jason Architecture for Programming Embedded Robotic Agents. In: Workshop on Engineering Multi-Agent Systems, 2016, Singapore. Proceedings of the Third International Workshop on Engineering Multi-Agent Systems (EMAS 2016), 2016.

REFERÊNCIAS BIBLIOGRÁFICA

- [Rao 1996] Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In: de Velde, W.V., Perram, J.W. (eds.) Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world. Lecture Notes in Artificial Intelligence, vol. 1038, pp. 42-55. Springer-Verlag, Secaucus. USA, 1996.
- [Stabile Jr. and Sichman, 2015] Stabile Jr., M.F., Sichman, J.S. Evaluating Perception Filters In BDI Jason Agents. In: 4th Brazilian Conference On Intelligent Systems, 2015.
- [Winikoff, 2005] Winikoff M. Jack intelligent agents: An industrial strength platform. Em Bordini R, Dastani M, Dix J, Fallah AS, Weiss G, editors. Multi-Agent Programming, volume 15 of Multiagent Systems, Artificial Societies, and Simulated Organizations, pages. 175-193. Springer US, 2005.
- [Wooldridge, 2000] Wooldridge, M. *Reasoning about rational agents. Intelligent robotics and autonomous agents. MIT Press*, 2000.
- [Wooldridge, 2009] Wooldridge M. An Introduction to MultiAgent Systems. John Wiley & Sons, 2009.
- [Zambonelli et al., 2001] Zambonelli F, Jennings NR, Omicini A, Wooldridge M. Agent-Oriented Software Engineering for Internet Applications. In: Omicini A, Zambonelli F, Klusch M, Tolksdorf R, editors. Coordination of Internet Agents. Springer Verlag; 2001. p.326-345, 2001

OBRIGADO!

souza.vdj@gmail.com

heder.to@gmail.com

pantoja@cefet-rj.br

viterbo@ic.uff.br

