
VINICIUS CELLANE ALVES - RGM:5032769326

EDUARDO MIOTO - RGM: 34604880

YURI FECCHIO LIMA - RGM: 32789602

PEDRO HENRIQUE LOPES SIQUEIRA – RGM: 33490252

WELLISON DA CRUZ BERTELLI 5033482004

**CST ADS.2° Semestre**

# Programação Web.

## Trabalho Carrinho Arduíno Bluetooth - Bônus: Testes Unitários de Software.

*Utilizando o aplicativo Android Dabble como interface de comunicação Homem-Máquina.*

Trabalho apresentado ao curso de Análise e Desenvolvimento de Sistemas na Disciplina de Programação Web sob supervisão do Profº Sidnei de Andrade como requisito parcial para obtenção de nota semestral da prova A1.

**Salto**
**2023**

## Código do carrinho, explicação detalhada sobre a Implementação abaixo dele:

**Testes Unitários no mesmo código fonte pois ele é integrado e pode ser monitorado tudo pelo aplicativo android Dabble, ou seja, não é necessário manter o Arduíno conectado no computador via USB para visualizar os resultados dos testes pela janela serial do ArduinoIDE.**

```
#include <ArduinoUnit.h>
#define CUSTOM_SETTINGS
#define INCLUDE_GAMEPAD_MODULE
#define INCLUDE_TERMINAL_MODULE
#include <Dabble.h>
#define INT1 2
#define INT2 3
#define INT3 4
#define INT4 5
class PrinterWrapperAndPluggerArduinoUnitToTerminalDabble : public Print {
public:
          String buffer;
          size_t write(uint8_t c) override {
                    buffer += (char)c;
                    return 1; // Indicando que um byte foi escrito
          }
          void sendStringToTerminalDabble() {
                    Terminal.print(buffer);
                    buffer = ""; // Limpa o buffer após o envio
          }
};
PrinterWrapperAndPluggerArduinoUnitToTerminalDabble
myPrintWrapper_pluggerArduinoUnitStdOutputs_toTerminalDabbleBluetooth;
int count = 0;
int qtde_execution_all_tests_cases = 1;
String serialdata = "";
bool dataflag = 0;
void setup() {
          Serial.begin(250000);
          Dabble.begin(9600,7,6);
          Test::out = &myPrintWrapper_pluggerArduinoUnitStdOutputs_toTerminalDabbleBluetooth;
          pinMode(INT1,OUTPUT);
          pinMode(INT2,OUTPUT);
          pinMode(INT3,OUTPUT);
          pinMode(INT4,OUTPUT);
          stop();
}
void loop() {
          Dabble.processInput();
          while (Serial.available() != 0) {
                    serialdata = String(serialdata + char(Serial.read()));
                    dataflag = 1;
          }
          if (dataflag == 1){
                    Terminal.print(serialdata);
                    serialdata = "";
                    dataflag = 0;
          }
          if(Terminal.available()){
                    while (Terminal.available() != 0) {
                              char command = (char) Terminal.read();
                              switch ((char) command) {
                                        case 't':
                                                  run_all_unit_test();
// Limpa as variáveis internas de controle do ArduinoUnit,
// para conseguirmos executar quantas vezes desejarmos os mesmos testes unitários executados // anteriormente, método
presente na versão mais recente dos últimos commits da lib.
                                                  Test::resetDoneTests();
                                                  break;
                                        default:
                                                  stop();
                              }
                    }
          }
          if(GamePad.isUpPressed()) {
                    forward();
          }
          else if(GamePad.isDownPressed()) {
                    backward();
          }
          else if(GamePad.isLeftPressed()) {
                    left();
          }
          else if(GamePad.isRightPressed()) {
                    right();
          }
          else {
                    stop();
          }
}
void forward() {
          digitalWrite(INT1,HIGH);
          digitalWrite(INT2,HIGH);
          digitalWrite(INT3,LOW);
          digitalWrite(INT4,LOW);
```

```
}
void backward() {
          digitalWrite(INT1,LOW);
          digitalWrite(INT2,LOW);
          digitalWrite(INT3,HIGH);
          digitalWrite(INT4,HIGH);
}
void left() {
          digitalWrite(INT1,LOW);
          digitalWrite(INT2,HIGH);
          digitalWrite(INT3,HIGH);
          digitalWrite(INT4,LOW);
}
void right() {
          digitalWrite(INT1,HIGH);
          digitalWrite(INT2,LOW);
          digitalWrite(INT3,LOW);
          digitalWrite(INT4,HIGH);
}
void stop() {
          digitalWrite(INT1,LOW);
          digitalWrite(INT2,LOW);
          digitalWrite(INT3,LOW);digitalWrite(INT4,LOW);
}
void sendDabbleMessage(String message) {
          Terminal.println(message);
}
void run_all_unit_test() {
          while (count <= qtde_execution_all_tests_cases){
                    sendDabbleMessage("Teste Número: "+String(count+1));
                    sendDabbleMessage("Iniciando bateria de testes...");
                    Test::run();
                    sendDabbleMessage("Finalizando bateria de testes...");
                    count++;
          }
          sendDabbleMessage("Generating Summary...");
          myPrintWrapper_pluggerArduinoUnitStdOutputs_toTerminalDabbleBluetooth.sendStr
          ingToTerminalDabble();
          count = 0;
}
// Units Tests cases for LEFT FUNCTION (PARA ESQUERDA) HARDWARE LEVEL
test(ok) {
          // Para FRENTE em cenário de sucesso
          sendDabbleMessage("- WHEN forward is successful THEN returns OK");
          forward();
          assertEqual(digitalRead(INT1), HIGH);
          assertEqual(digitalRead(INT2), HIGH);
          assertEqual(digitalRead(INT3), LOW);
          assertEqual(digitalRead(INT4), LOW);
          sendDabbleMessage("- Finalizou teste SUCCESSFUL no forward...");
          delay(800);
          // Para TRÁS em cenário de sucesso
          sendDabbleMessage("- WHEN backward is successful THEN returns OK");
          backward();
          assertEqual(digitalRead(INT1), LOW);
          assertEqual(digitalRead(INT2), LOW);
          assertEqual(digitalRead(INT3), HIGH);
          assertEqual(digitalRead(INT4), HIGH);
          sendDabbleMessage("- Finalizou teste SUCCESSFUL no backward...");
          delay(800);
          // Para ESQUERDA em cenário de sucesso
          sendDabbleMessage("- WHEN left is successful THEN returns OK");
          left();
          assertEqual(digitalRead(INT1), LOW);
          assertEqual(digitalRead(INT2), HIGH);
          assertEqual(digitalRead(INT3), HIGH);assertEqual(digitalRead(INT4), LOW);
          sendDabbleMessage("- Finalizou teste SUCCESSFUL no left...");
          delay(800);
          // Para DIREITA em cenário de sucesso
          sendDabbleMessage("- WHEN right is successful THEN returns OK");
          right();
          assertEqual(digitalRead(INT1), HIGH);
          assertEqual(digitalRead(INT2), LOW);
          assertEqual(digitalRead(INT3), LOW);
          assertEqual(digitalRead(INT4), HIGH);
          sendDabbleMessage("- Finalizou teste SUCCESSFUL no right...");
          delay(800);
          stop();
          }
test(bad) {
          // Para FRENTE, não pode ser igual a nenhuma outras direções
          sendDabbleMessage("- WHEN forward is error THEN returns BAD");
          forward();
          // Não pode ser igual aos setters para trás
          assertNotEqual(digitalRead(INT1), LOW);
          assertNotEqual(digitalRead(INT2), LOW);
          assertNotEqual(digitalRead(INT3), HIGH);
          assertNotEqual(digitalRead(INT4), HIGH);
          // Também Não pode ser igual aos setters para esquerda e direita, mas TESTAR
          ESSES CASOS estava sobrecarregando o arduino...
          sendDabbleMessage("- Finalizou teste ERROR no forward...");
          delay(800);
          // ======================================================
          // Para TRÁS, não pode ser igual a nenhuma outras direções
          sendDabbleMessage("- WHEN backward is error THEN returns BAD");
          backward();
          // Não pode ser igual aos setters para frente
          assertNotEqual(digitalRead(INT1), HIGH);
          assertNotEqual(digitalRead(INT2), HIGH);
          assertNotEqual(digitalRead(INT3), LOW);
          assertNotEqual(digitalRead(INT4), LOW);
```

```
        // Também Não pode ser igual aos setters para esquerda e direita, mas TESTAR
        ESSES CASOS estava sobrecarregando o arduino...
        sendDabbleMessage("- Finalizou teste ERROR no backward...");
        delay(800);
        // ========================================================
        // Para ESQUERDA, não pode ser igual a nenhuma outras direções
        sendDabbleMessage("- WHEN left is error THEN returns BAD");
        left();// Não pode ser igual aos setters para direita
        assertNotEqual(digitalRead(INT1), HIGH);
        assertNotEqual(digitalRead(INT2), LOW);
        assertNotEqual(digitalRead(INT3), LOW);
        assertNotEqual(digitalRead(INT4), HIGH);
        // Também Não pode ser igual aos setters para frente e trás, mas TESTAR ESSES
        CASOS estava sobrecarregando o arduino...
        sendDabbleMessage("- Finalizou teste ERROR no left...");
        delay(800);
        // ========================================================
        // Para DIREITA, não pode ser igual a nenhuma outras direções
        sendDabbleMessage("- WHEN right is error THEN returns BAD");
        right();
        // Não pode ser igual aos setters para esquerda
        assertNotEqual(digitalRead(INT1), LOW);
        assertNotEqual(digitalRead(INT2), HIGH);
        assertNotEqual(digitalRead(INT3), HIGH);
        assertNotEqual(digitalRead(INT4), LOW);
        // Também Não pode ser igual aos setters para frente e para trás, mas TESTAR
        ESSES CASOS estava sobrecarregando o arduino...
        sendDabbleMessage("- Finalizou teste ERROR no right...");
        delay(800);
        stop();
}
```

## Overview

Utilizamos o aplicativo Android Dabble como interface de comunicação Homem-Máquina.

Apesar de utilizarmos como base o seguinte projeto do makerhero, muitas mudanças foram necessárias para funcionar corretamente:

https://www.makerhero.com/blog/como-fazer-um-carrinho-de-controle-remoto-simples-com-bluetooth/

- Carrinho controlado via GamePad Module do aplicativo android Dabble.
- Bateria de testes unitários disparados e monitorados via Terminal Module do mesmo aplicativo, mencionado acima.

## Primeiros problemas

Lógica não condiz com esquema elétrico apresentado, conexões dos motores das rodas NÃO correspondem ao código fornecido pelo projeto base do makerhero.

Direções estavam trocadas, a funcionalidade `"para frente"` indo `"para esquerda"`, e para os lados também incorretos.

**Solução:** Bastando extrair o bloco `"para esquerda"` e plugar no bloco `"para frente"` e assim por diante, já foi possível resolver.

## Biblioteca para testes unitários de software ArduinoUnit

**Documentação oficial:** https://github.com/mmurdoch/arduinounit

Biblioteca do Arduino especializada em testes unitários (bem similar ao JUnit do Java), por meio dos Assertions.

Primeiro devemos definir os métodos de testes, passando como parâmetro qual é o tipo de Assertion que estamos realizando.

**Exemplo de teste unitário para o cenário simples `"When forward is successful then returns OK"`:**

```
test(ok) {
        Serial.println("- WHEN forward is successful THEN returns OK");
        forward(); // Método que vamos testar, "para frente".
        assertEqual(digitalRead(INT1, HIGH));
        assertEqual(digitalRead(INT2, HIGH));
        assertEqual(digitalRead(INT3, LOW));
        assertEqual(digitalRead(INT4, LOW));
        Serial.println("- Finalizou teste SUCCESSFUL no forward...");
}
```

Após definir os métodos de testes, deve-se executar a chamada para o método estático `Test::run();`

Que inicia a bateria de testes para TODOS os métodos de `test(ok | bad...){...}` definidos.

## Terminal Module Dabble

**Documentação oficial:** https://ai.thestempedia.com/docs/dabble-app/terminal-module/

Pelo Terminal Module do aplicativo android Dabble é possível realizar trocas de mensagem texto entre o celular e o Arduino via bluetooth.

E de acordo com essas mensagens podemos realizar diversas ações no Arduino.

Neste caso, quando enviado um `"t"` a bateria de testes unitários é iniciada.

A utilização é bem simples, basta instalar a biblioteca do Dabble pelo gerenciador de dependências da IDE.

Depois, é só realizar o include do `#include <Dabble.h>` e `#define INCLUDE_TERMINAL_MODULE`

Após configurações de ambiente finalizadas, basta utilizar blocos `switch-case` ou similar para aplicar lógicas quando `"t"` for recebido.

**Exemplo de utilização do Terminal Module, quando receber `"t"` bateria de testes unitários é iniciada:**

```
String serialdata = "";
bool dataflag = 0;
void loop() {
        Dabble.processInput();
        while (Serial.available() != 0) {
                serialdata = String(serialdata + char(Serial.read()));
                dataflag = 1;
        }
        if (dataflag == 1){
                Terminal.print(serialdata);
                serialdata = "";
                dataflag = 0;
        }
        if(Terminal.available()){
                while (Terminal.available() != 0) {
                        char command = (char) Terminal.read();
                        switch ((char) command) {
                                case 't':
                                        run_all_unit_test();
                                        break;
                                default:
                                        stop();
                        }
                }
        }
}
```

## ArduinoUnit e Polimorfismo

**Documentação oficial:** https://ai.thestempedia.com/docs/dabble-app/terminal-module/

A saída padrão dos resultados dos testes para monitoramentos e feedbacks sobre eles, é por meio do stdout `Serial.print...()` nativo do arduíno.

Ou seja, para debbugar e verificar esses resultados, deve-se estar com o arduíno conectado no computador via cabo USB.

E visualizar por meio da janela serial do ArduínoIDE.

Porém isso é bem chato e desgastante, então resolvi me desafiar ainda mais.

Estamos com uma conexão Bluetooth estabelecida entre o celular e o carrinho.

Então podemos utilizar o mesmo ambiente como interface de comunicação homem-máquina, para monitoramento dos testes: `Aplicativo Android Dabble - Terminal Module.`

Com uso do polimorfismo, podemos implementar uma classe compatível com `Print` herdando dele.

Pois "por debaixo dos panos" a biblioteca ArduinoUnit utiliza os métodos dele para realizar impressões print, uma vez que ele é a super classe do `Serial`

Então basta sobrescrever o método `write()` do `Print`, pois ele é executado por todos os métodos de impressões do `Serial.print...()`

**Exemplo de classe compatível com Print utilizando polimorfismo por meio de herança:**

```
class PrintAdapterArduinoUnitToTerminalDabble : public Print {
    public:
    String buffer;

    size_t write(uint8_t c) override {
        buffer += (char) c;
        return 1;
    }

    void sendMsgToTerminalDabble() {
        Terminal.print(buffer);
        buffer = ""; // Limpa o buffer após o envio.
    }

    void sendMsgToTerminalDabble(String message) {
        Terminal.println(message);
    }
}
PrintAdapterArduinoUnitToTerminalDabble
myPrintAdapter_arduinoUnitStdout_toTerminalDabbleBluetooth;
```

E depois, alterar o ponteiro padrão do **Test::out = &Serial;** para uma instância da classe compatível com **Print** criado por nós anteriormente.

**Exemplo de como alterar o ponteiro, não precisa ser necessáriamente neste bloco, mas deve ser executado antes da bateria de testes unitários:**

```
void setup() {
    ...
    Test::out = &myPrintAdapter_arduinoUnitStdout_toTerminalDabbleBluetooth;
    ...
}
```

Desta forma, podemos personalizar as regras de saída serial, redirecionando todas as impressões para outros módulos do projeto.

Neste caso, redirecionamos para o Terminal Module, que é outra funcionalidade do aplicativo Android Dabble.

Executamos essa ação após finalizar toda a bateria de testes unitários, enviando o buffer acumulado.

**Exemplo de chamada para execução da bateria de testes, e após finalizado, envia o feedback dos testes acumulados no buffer:**

```
int count = 0;
int qtde_execution_all_tests_cases = 1;

void run_all_unit_test() {
    while (count >= qtde_execution_all_tests_cases) {
        sendMsgToTerminalDabble("Teste Número: "+String(count+1));
        sendMsgToTerminalDabble("Iniciando bateria de testes...");
        Test::run();
        sendMsgToTerminalDabble("Finalizando bateria de testes...");
        count++;
    }
sendMsgToTerminalDabble("Generating Summary...");

// Após finalizado, envia buffer acumulado do `Test::out.print()` para Terminal Module:
sendMsgToTerminalDabble();
count = 0;

}
```

## Ajuste fino no ArduinoUnit

Após finalizar toda a bateria de testes unitários, variáveis de controles internas da biblioteca **ArduinoUnit** são definidas como **"Já executado".**

Isso implica que a bateria de testes só poderá ser executada UMA única vez, a cada compilação das instruções no Arduino.

Isso ocorre para não ficar executando o mesmo método de teste repetidas vezes, MAS isso também ocorre quando os parâmetros dos testes são diferentes.

E isso é ruim, pois e se quisermos executar a mesma função, porém com diferentes parâmetros?

Na versão LTS da main instalado via gerenciador de dependências do ArduinoIDE isso não é possível, então deve-se instalar o seguinte commit:

https://github.com/mmurdoch/arduinounit/commit/c96d03d405dea4be713768287070e64fa676ccbb

Após a execução por completo dos testes **Test:run()** deve-se executar o resetter **resetDoneTests()** para limpar as variáveis internas de controle do ArduinoUnit.

Desta forma, podemos enviar o comando **"t"** várias vezes pelo aplicativo android Dabble que os testes serão executados quantas vezes quisermos.

## Resultado


SUPER
HIGH-SPEED
REALITY
SPORT