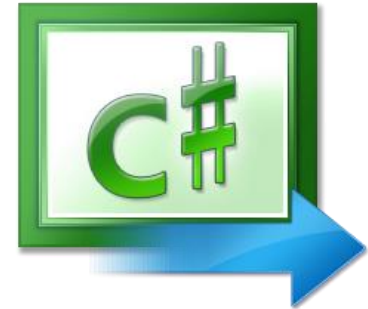


# Multithreading & Comunicação Serial usando Arduino e C#

**Marcelo Vinícius Cysneiros Aragão**  
<http://www.contactify.com/bf737>

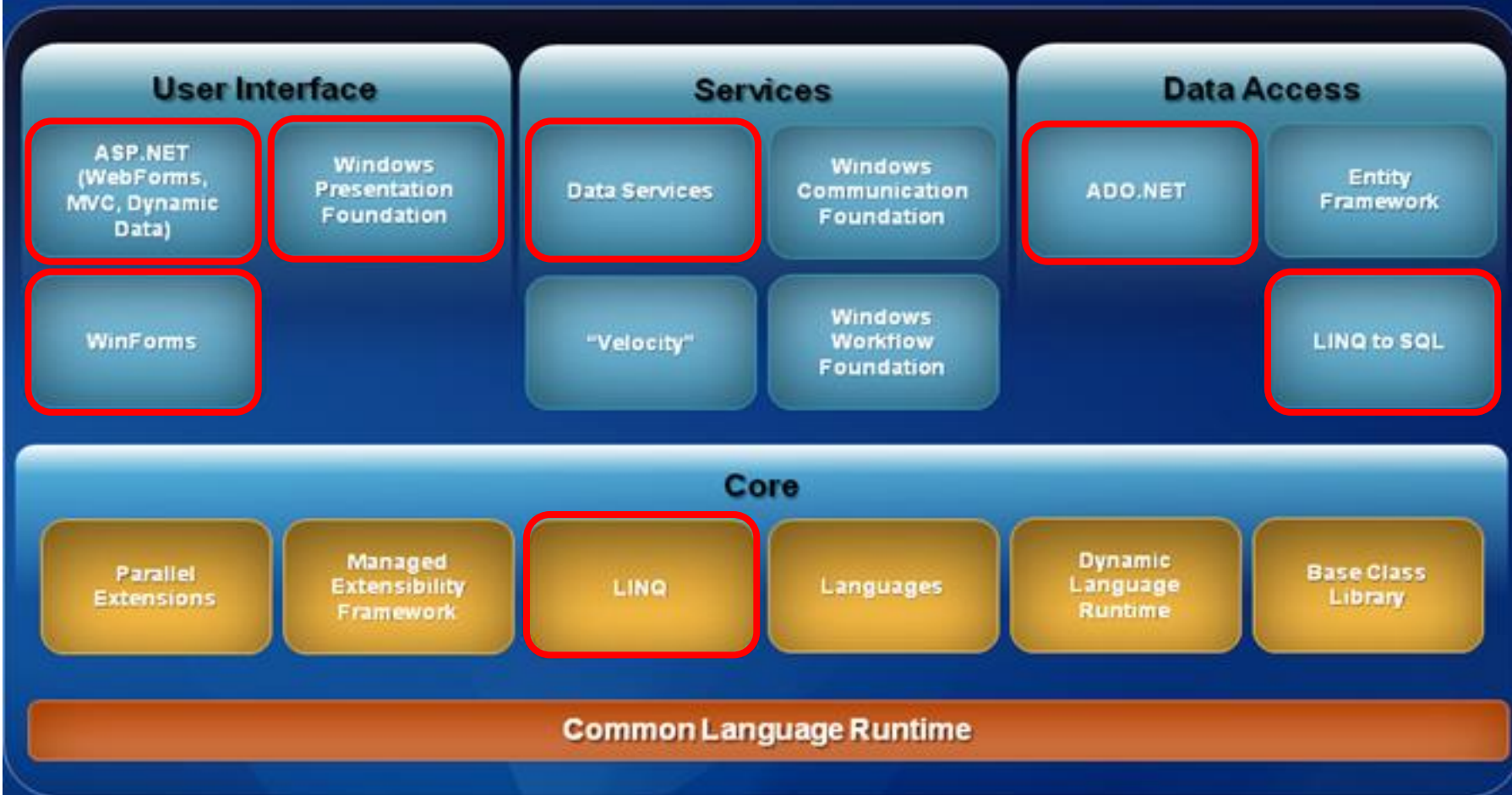
# Introdução: C# e .NET Framework

- ❑ Linguagem OO fortemente tipada, baseada em C++, VB, Object Pascal e Java.
- ❑ Desenvolvida em 2001 pela Microsoft como parte da Plataforma .NET (atualmente, está na versão 4.0).



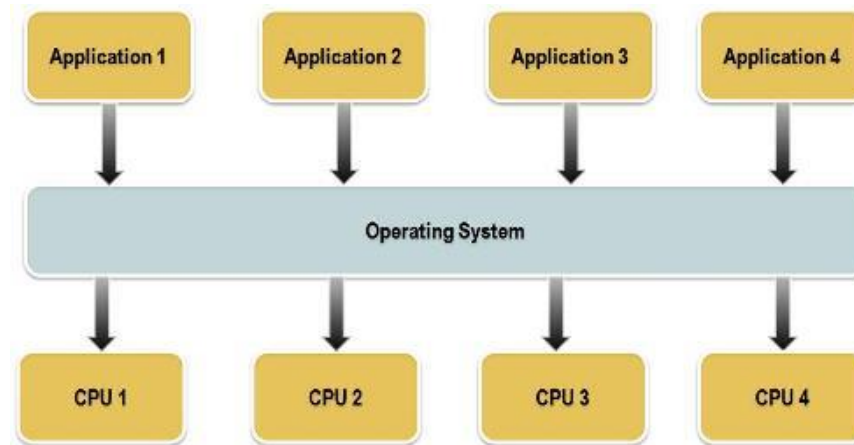
- ❑ Plataforma única para desenvolvimento e execução de sistemas e aplicações (ASP.NET, WindowsForms, WPF, ADO.NET, LINQ).
- ❑ Portabilidade: qualquer código gerado para **.NET** pode ser executado em qualquer dispositivo que possua um framework de tal plataforma.
- ❑ .NET Framework 4.0: retrocompatibilidade, Background Garbage Collection, suporte a dispositivos com Multitouch e boa integração com Windows 7.
- ❑ Curiosidade: DotGnu e Mono Project (ferramentas compatíveis com plataforma .NET; pode ser executado em Linux, BSD, UNIX, Mac OS X, Solaris e Windows).

# .NET Framework 4.0



# Introdução: Multithreading

- ❑ Definição: “forma de um processo dividir a si mesmo em duas ou mais tarefas que podem ser executadas concorrentemente.”
- ❑ O suporte à thread pode ser fornecido pelo próprio sistema operativo ou implementada através de uma biblioteca de uma determinada linguagem.



- ❑ Em *hardwares* com uma única CPU, cada thread é processada de forma aparentemente simultânea devido à uma mudança rápida de contexto (sensação de paralelismo)
- ❑ Em *hardwares* com múltiplos CPUs ou multi-cores, as threads são realizadas realmente de forma simultânea.

# Introdução : BackgroundWorker e lock{}

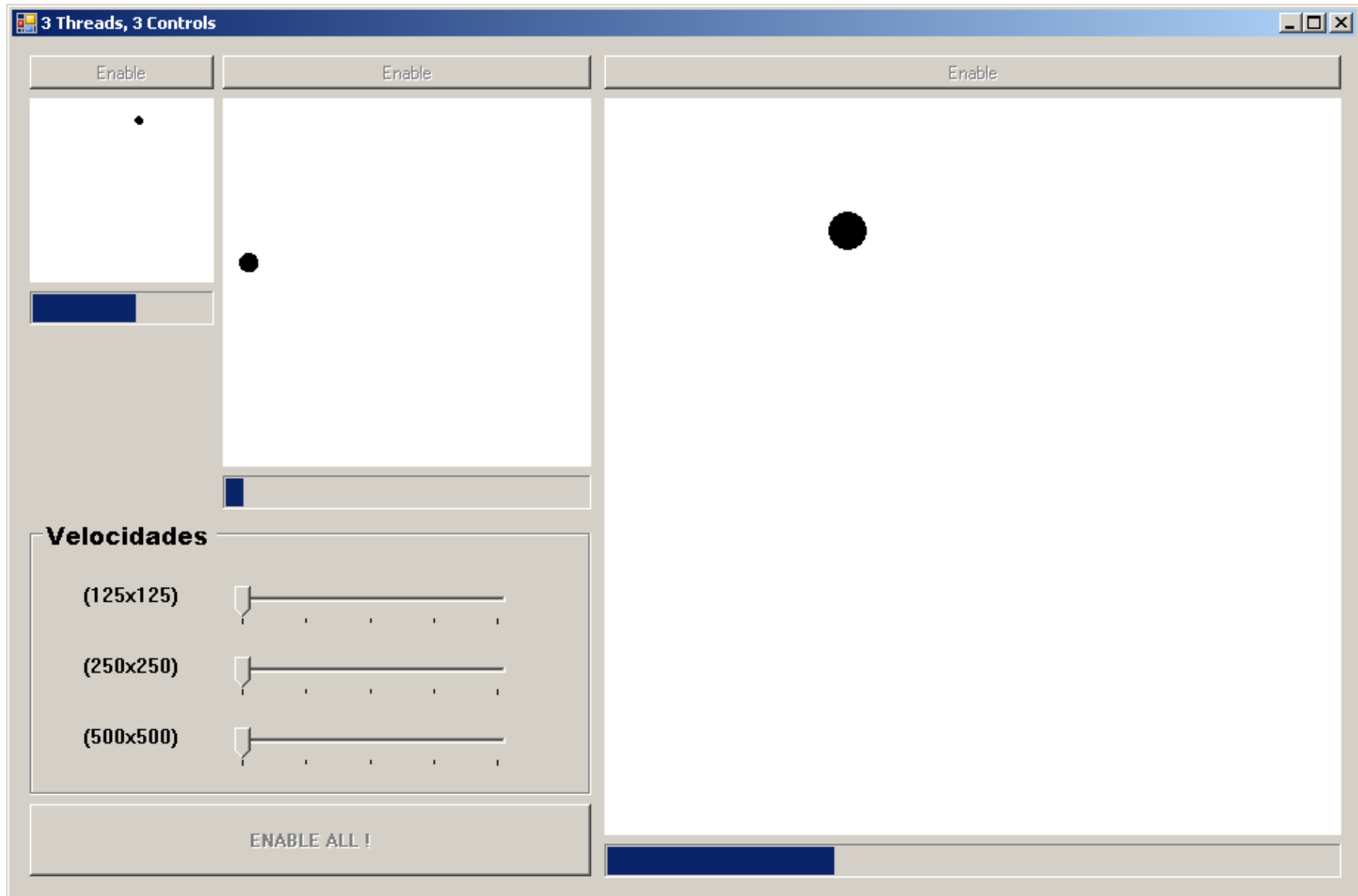
- “A classe `BackgroundWorker` provê um modo fácil de executar operações custosas/demoradas numa “thread em background”. Ela também torna possível checar o estado da operação, inclusive permitindo cancelá-la.”
- “Quando a classe `BackgroundWorker` é utilizada, também é possível indicar (“reportar”) o progresso, cancelamento e “completude” da operação na própria interface do usuário. Exemplo: verifica-se se a operação em background foi completada ou cancelada e mostra uma mensagem ao usuário, com o status da operação.”

[http://msdn.microsoft.com/fr-fr/library/cc221403\(v=vs.95\).aspx](http://msdn.microsoft.com/fr-fr/library/cc221403(v=vs.95).aspx), MSDN, adaptado

“A palavra-chave `lock` (bloqueio de execução) garante que um thread não entre em uma seção crítica do código enquanto outro thread está na seção crítica. Se outro segmento tenta digitar um código bloqueado, esperará, para bloquear, até que o objeto seja liberado.”

<http://msdn.microsoft.com/pt-br/library/vstudio/c5kehkcz.aspx>, MSDN, adaptado

# Exemplo 1



3 Threads, 3 Controls e ReportProgress( ) com ProgressBar

# Exemplo 1: BackgroundWorker

```
private System.ComponentModel.BackgroundWorker backgroundWorker0;
```

→ Declaração do BackgroundWorker

```
private void InitializeComponent()
{
    this.backgroundWorker0 = new System.ComponentModel.BackgroundWorker();
    //
    // backgroundWorker0
    //
    this.backgroundWorker0.WorkerReportsProgress = true;
    this.backgroundWorker0.WorkerSupportsCancellation = true;
    this.backgroundWorker0.DoWork += new System.ComponentModel.DoWorkEventHandler(this.backgroundWorker0_DoWork);
    this.backgroundWorker0.ProgressChanged += new System.ComponentModel.ProgressChangedEventHandler(this.backgroundWorker0_ProgressChanged);
}
```

→ Inicialização

Ativa suporte a cancelamento

Define o método do evento DoWork()

Define o método do evento ProgressChanged()

# Exemplo 1: DoWork & ProgressChanged

```
private void backgroundWorker0_DoWork(object sender, DoWorkEventArgs e)
{
    while (true)
    {
        CalculateBounce(0, pictureBox0, 5.0f);
        backgroundWorker0.ReportProgress((int)(pt[0].X / 1.25));
        System.Threading.Thread.Sleep(10);
    }
}
```

Calcula trajetória da bolinha

→ Reporta o progresso (neste caso, posição horizontal)

Sleep de 10ms

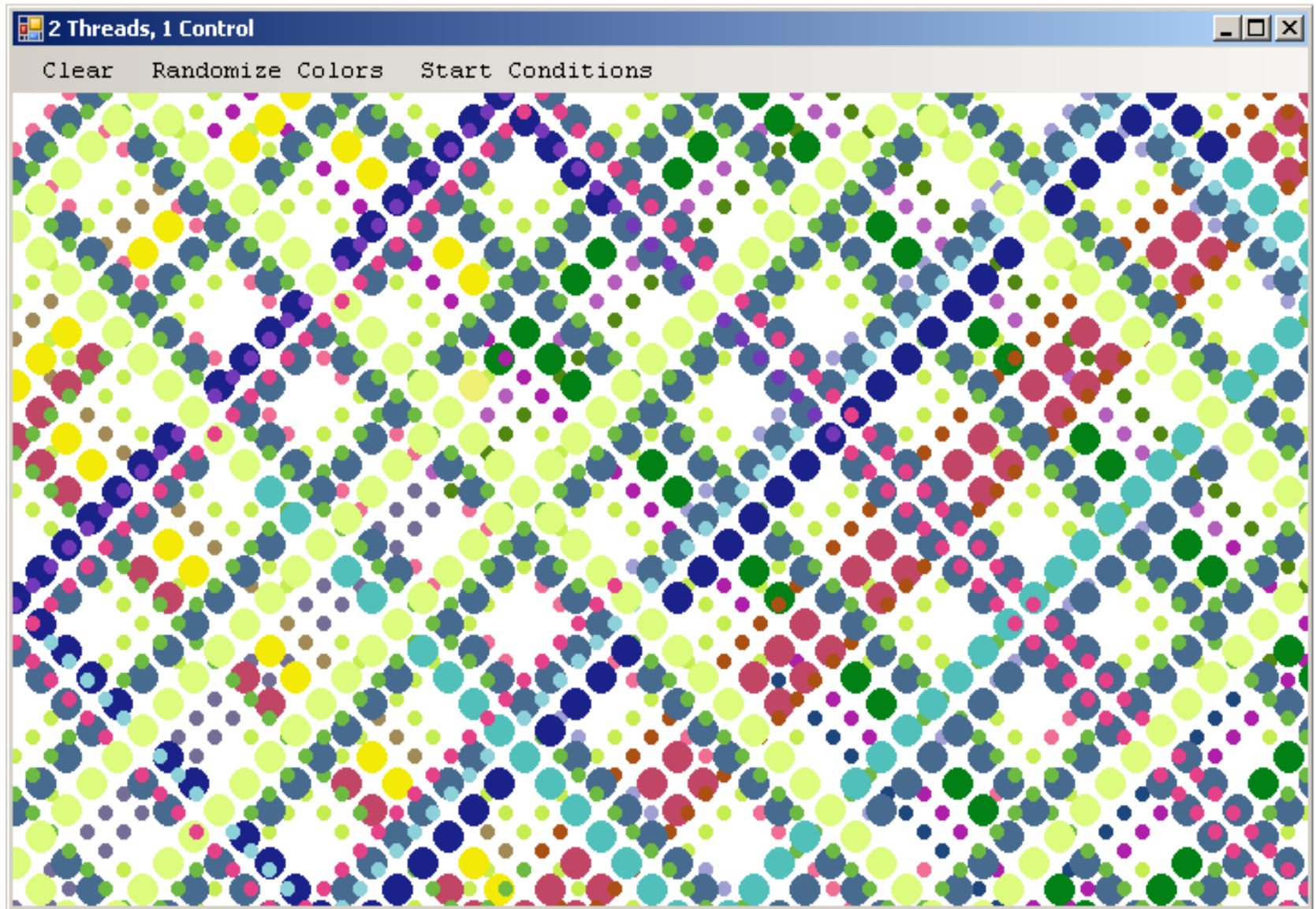
```
private void backgroundWorker0_ProgressChanged(object sender, ProgressChangedEventArgs e)
{
    try { progressBar0.Value = e.ProgressPercentage; }
    catch (Exception) { }
}
```

→ Evento de “progresso alterado” do BackgroundWorker:

Seta a posição da barra de progresso como o progresso reportado



# Exemplo 2



Exemplo 2: 2 Threads, 1 Control e lock{ }

# Exemplo 2: lock{}

```
private void backgroundWorker1_DoWork(object sender, DoWorkEventArgs e)
{
    while (true)
    {
        CalculateBounce(0);

        try
        {
            lock (lockObject)
            {
                if (pictureBox1.Image == null)
                    pictureBox1.Image = new Bitmap(pictureBox1.Width, pictureBox1.Height);

                Graphics g = Graphics.FromImage(pictureBox1.Image);
                g.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.HighSpeed;
                g.DrawEllipse(balls[0].pen, balls[0].X, balls[0].Y, balls[0].radix, balls[0].radix);
                g.FillEllipse(balls[0].brush, balls[0].X, balls[0].Y, balls[0].radix, balls[0].radix);
                pictureBox1.Invalidate();
                System.Threading.Thread.Sleep(delay);
            }
        }
        catch (Exception) { }
    }
}
```

Calcula trajetória da bolinha #1

“Trava” o objeto *lockObject* (entra em seção crítica do código)

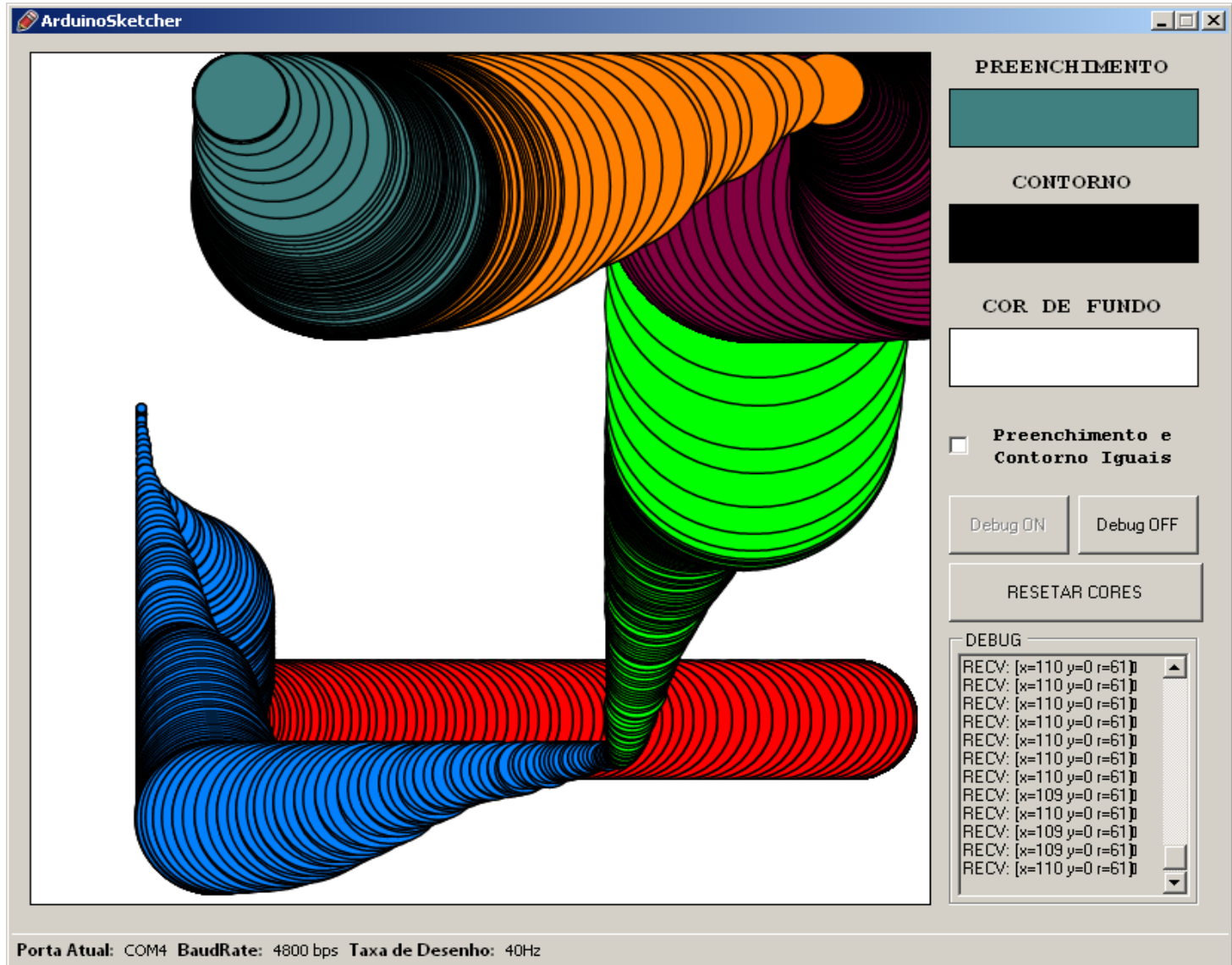
Desenho é custoso, por isso está na seção crítica

Desenha a bolinha #1 (modo HighSpeed) em sua nova posição

Sleep para aliviar o processamento

Sai da seção crítica

# Exemplos 3



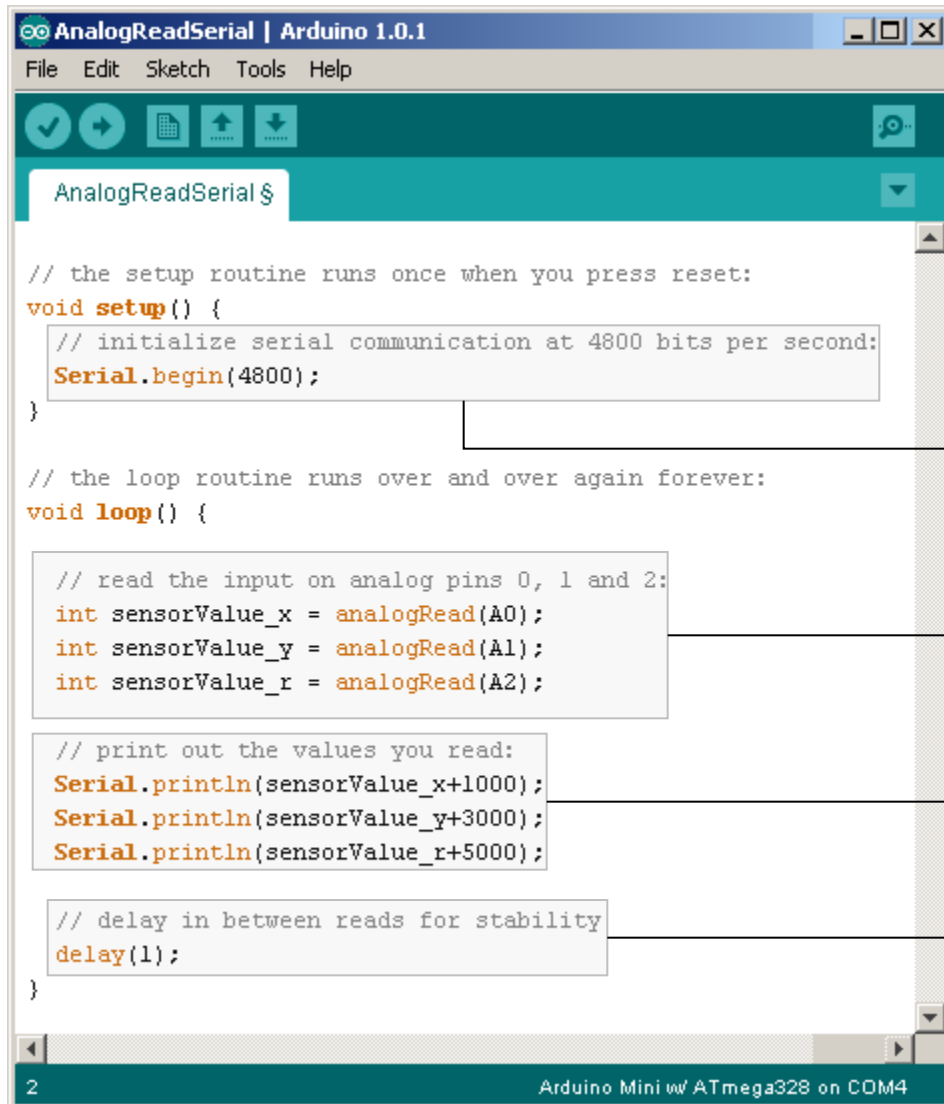
Exemplo 3: 2 Threads e Arduino (serial-usb)

# Introdução: Arduino

- ❑ Plataforma de prototipagem eletrônica de hardware livre, projetada com um microcontrolador Atmel AVR de placa única
- ❑ Oferece suporte de entrada/saída embutido, linguagem de programação padrão baseada, essencialmente, C/C++.
- ❑ Objetivo: criar ferramentas acessíveis e de baixo custo à amadores e desenvolvedores (principalmente para aqueles que não teriam alcance aos controladores mais sofisticados e a ferramentas mais complicadas).



# Exemplo 3: Código Arduino



```
// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 4800 bits per second:
  Serial.begin(4800);
}

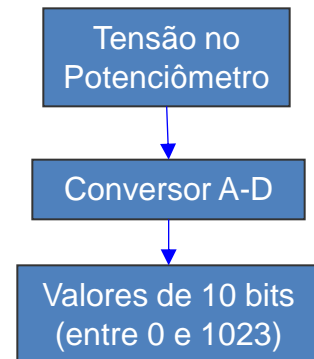
// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pins 0, 1 and 2:
  int sensorValue_x = analogRead(A0);
  int sensorValue_y = analogRead(A1);
  int sensorValue_r = analogRead(A2);

  // print out the values you read:
  Serial.println(sensorValue_x+1000);
  Serial.println(sensorValue_y+3000);
  Serial.println(sensorValue_r+5000);

  // delay in between reads for stability
  delay(1);
}
```

Inicia a comunicação serial a uma taxa de comunicação (baud rate) de 4800bps

3 inteiros recebem o valor das leituras dos pinos analógicos A0, A1 e A2.



Valores lidos são “printados” na de forma escalonada (offset), para posterior identificação no programa.

Delay de 1ms para “aliviar” o processador

# Exemplo 3: BGWorker e SerialPort

```
System.ComponentModel.BackgroundWorker backgroundWorkerPosicao;  
System.ComponentModel.BackgroundWorker backgroundWorkerCores;  
System.IO.Ports.SerialPort serialPort;
```

Declaração dos BackgroundWorker  
Declaração da Porta Serial

```
//inicializa porta serial  
serialPort = new SerialPort();  
try {  
    serialPort.BaudRate = 4800;  
    serialPort.PortName = portaEscolhida;  
    serialPort.Open();  
}  
catch (System.IO.IOException)  
{  
    MessageBox.Show("Porta inválida selecionada.\nSaindo do programa.", "Erro");  
    System.Environment.Exit(0);  
}  
catch (ArgumentException)  
{  
    MessageBox.Show("Janela de seleção de porta foi fechada.\nSaindo do programa.", "Aviso");  
    System.Environment.Exit(0);  
}
```

Inicialização da  
porta serial  
(4800bps, COM4)

Porta serial passa  
a receber dados

Tratamento de  
exceção para  
PortName inválido

```
//inicializa timer para refresh  
System.Timers.Timer t = new System.Timers.Timer();  
t.Elapsed += new ElapsedEventHandler(timer_Tick);  
t.Interval = 25;  
t.Start();
```

Timer que “emite” um evento  
timer\_Tick a cada 25ms

```
//evento que acontece a cada tick do timer  
public void timer_Tick (object sender, ElapsedEventArgs e) {  
    drawCircle(propCirculo.X,propCirculo.Y,propCirculo.R);  
    if (debug) AtualizaCaixaDebug();  
    System.Threading.Thread.Sleep(10);  
}
```

Evento do timer: redesenha a imagem a cada 25ms,  
atualiza linha de debug e faz um sleep de 10ms



# Exemplo 3: BGWorker e SerialPort

```
public class PropCirculo
{
    public int x;
    public int y;
    public int r;
}
```

Classe que representa as propriedades do círculo:

- Posição no eixo X (horizontal)
- Posição no eixo Y (vertical)
- Raio

```
//inicia Background worker de comunicação serial
propCirculo = new PropCirculo();
backgroundWorkerPosicao.RunWorkerAsync(propCirculo);
```

Inicializa a variável com propriedades do círculo  
Inicia o BackgroundWorker de Posição, **passando propCirculo como parâmetro**

```
void BackgroundWorkerPosicaoDoWork(object sender, System.ComponentModel.DoWorkEventArgs e)
{
    while (true) {
        int val;
        string s = serialPort.ReadLine();
        if (int.TryParse(s, out val)) {
            if (val < 3000) {
                propCirculo.X = (val-1000)/2;
            }
            else if (val >= 3000 && val < 5000) {
                propCirculo.Y = (val-3000)/2;
            }
            else {
                propCirculo.R = (val-5000)/5;
            }
            if(debug)
                debugString = String.Format("RECV: [x={0} y={1} r={2}]\n",
                                            propCirculo.X, propCirculo.Y, propCirculo.R);
            System.Threading.Thread.Sleep(1);
        }
    }
}
```

Loop infinito, que, executado em background pelo BackgroundWorker de Posição:

- Reconhece à qual dado (x, y, raio) pertence a informação lida da porta serial, de acordo com cada *offset*.
- Atualiza propCirculo
- Atualiza string de debug
- Sleep de 1ms.

# Exemplo 3: BGWorker e SerialPort

```
void BackgroundWorkerCoresDoWork(object sender, System.ComponentModel.DoWorkEventArgs e)
{
    // chosenOption define qual cor deve ser alterada
    switch (chosenOption)
    {
        case 1: // mudanca da cor do preenchimento
            if (colorDialog.ShowDialog() == DialogResult.OK)
            {
                brushGlobal = new SolidBrush(colorDialog.Color);
                btnCorPreenchimento.BackColor = colorDialog.Color;
            }
            break;

        case 2: // mudanca da cor de contorno
            if (colorDialog.ShowDialog() == DialogResult.OK)
            {
                penGlobal.Color = colorDialog.Color;
                btnCorContorno.BackColor = colorDialog.Color;
            }
            break;

        case 3: // mudanca da cor de fundo
            if (colorDialog.ShowDialog() == DialogResult.OK)
            {
                corDeFundo = colorDialog.Color;
                btnCorDeFundo.BackColor = colorDialog.Color;
                pictureBox1.Image = null;
            }
            break;
    }
}
```

```
void btnPreenchimentoClick(object sender, System.EventArgs e)
{
    chosenOption = 1;
    try { backgroundWorkerCores.RunWorkerAsync(brushGlobal); }
    catch (InvalidOperationException) { }
}
```

Definição “DoWork” do BackgroundWorker de Cores:

- Através da variável “chosenOption”, identifica se a mudança de cor é referente ao preenchimento, contorno ou cor de fundo.
- Atualiza a nova cor.
- No caso de alteração de cor de fundo, “reinicia o desenho”, com a nova cor de fundo.

Um dos botões que utiliza o BackgroundWorker de Cores (preenchimento = 1, contorno = 2, cor de fundo = 3).



# Conclusões

A linguagem C#, juntamente com a plataforma .NET, oferece métodos fáceis para criação de aplicações multithread, fazendo melhor uso dos recursos do processador.

A classe BackgroundWorker apresenta grande aplicabilidade e uso simplificado, inclusive em relação à “notificação” de progresso (usado em instaladores e patches, por exemplo).

Através da instrução *lock*, o desenvolvedor consegue definir regiões críticas no código, evitando problemas de *crossthreading*\*.

Por último, a interface entre o Arduino e a classe SerialPort nativa do C# simplificou bastante a comunicação com uma porta serial, proporcionando grande possibilidade de interação entre software e circuitos eletrônicos, como sensores, alarmes, etc.

