Sinalgo

Bruno Pereira

Universidade Federal de Minas Gerais

bruno.ps@dcc.ufmg.com

30 de agosto de 2016

Agenda

1. Introdução
   - Sinalgo
2. Primeros Passos com o Sinalgo
   - Ambiente de Desenvolvimento
   - Instalação e Documentação
3. Projetos no Sinalgo
   - Árvore de Diretório de um Projeto
   - Criando um Projeto
   - Implementação do Projeto
4. Exercícios
   - Exemplos do Sinalgo
   - Árvore de Roteamento

Introdução

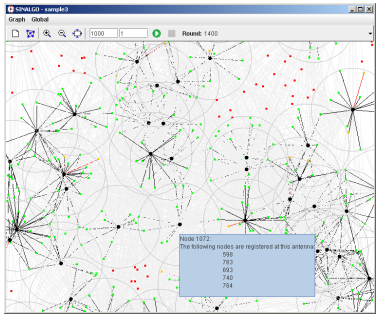
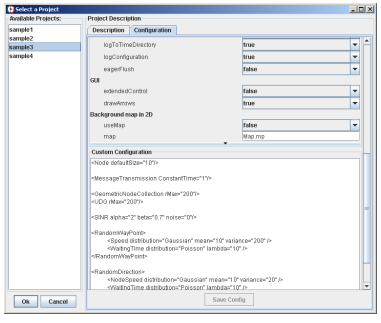
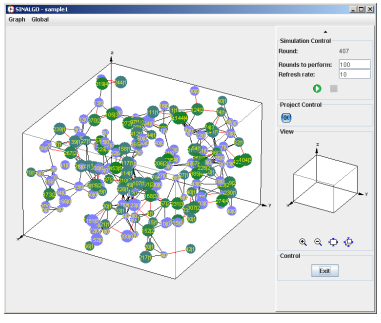
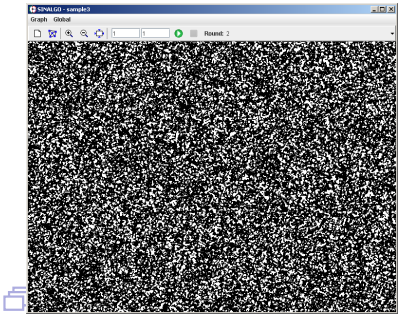
O que é?

- É uma ferramenta para testar e validar algoritmos de rede.
- Sinalgo abstrai de modo fácil e simples as camadas de enlace e física.
- Os nós enviam ou recebem mensagens unicast ou broadcast, reagem a mensagens recebidas, usam timers para escalonar ações, dentre outros.

Introdução

Características

1. Torna fácil o desenvolvimento de algoritmos pois é implementado em ~~*ctrl + space*~~ JAVA.
2. Alto desempenho, pois executa simulações com milhares de nós.
3. Suporte a 2D e 3D.
4. Simulação síncrona e assíncrona.
5. Independente de plataforma (JAVA).
6. Ambiente visual da simulação da rede.

Introdução

Fácil extensão (models)

- Mobilidade.
- Conectividade.
- Distribuição.
- Confiabilidade.
- Transmissão de Mensagens.

Getting Started

Pré-requisitos

- Java 5.0 (J2SE 5.0 JDK) ou superior.
- Eclipse.

Versões

- Regular Release (recomendo).
- Toy Release.

Getting Started

Como instalar? (Regular Release com Eclipse)

1. Crie um projeto ('File' → 'New' → 'Project')
2. Escolha um projeto do tipo 'Java Project'.
3. Dê um nome de sua preferência (ex: "SinalgoAula").
4. Importe o Sinalgo Regular Release para seu Workspace.
   1. Extraia sinalgo-x.xx.x.zip para um diretório temporário
   2. Faça a cópia do conteúdo do Regular Release para o seu projeto
   3. Certifique-se de que o arquivo **.classpath** foi sobrescrito.
5. Verifique se Eclipse está configurado com Java 5.0 ou superior.
   1. ('Window' → 'Preferences'), 'Java' → 'Compiler'
   2. ('Window' → 'Preferences'), select 'Java' → 'Installed JREs'

- Mais detalhes... **Aqui**.

Getting Started

Como executar?

1. Clique com o botão direito na pasta "src" dentro da aba "Project Explorer" ou "Navigator" do Eclipse
2. Clique na opção "Run As" → "Java Application"
3. Na tela "Select Java Application", selecione a classe "Run".
4. Mais detalhes sobre **instalação** e **execução**.

Getting Started

Como executar?

1. Clique com o botão direito na pasta "src" dentro da aba "Project Explorer" ou "Navigator" do Eclipse
2. Clique na opção "Run As" → "Java Application"
3. Na tela "Select Java Application", selecione a classe "Run".
4. Mais detalhes sobre **instalação** e **execução**.

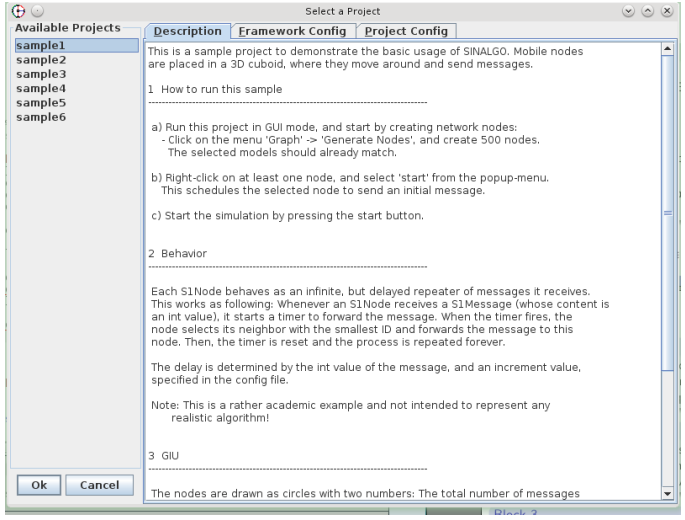


Figura: se tudo deu certo!

Getting Started

Documentação

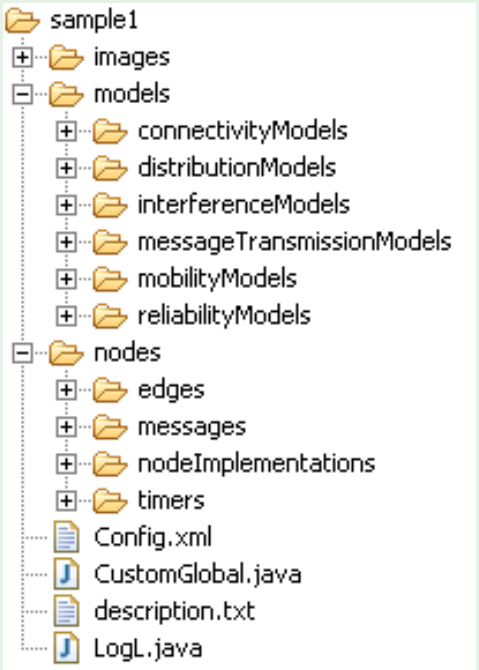
1. O código é bem documentado
   1. Os 6 exemplos podem ser utilizados como base
2. **doxygen**
3. **Sinalgo Site!**
4. Mais detalhes sobre **instalação** e **execução**.

```
    /**
     * The framework calls this method after starting the application and
     * before executing the first round.
     * <p>
     * This method may be used to perform any task that needs to be executed
     * before the simulation starts, e.g. initialize some datastructures.
     * <p>
     * By default, this method does nothing.
     */
    public void preRun() {
        // No implementation here! Add your code to the CustomGlobal.java
        // file in your project.
    }

    /**
     * The framework calls this method before each round. I.e. before the positions
     * of the nodes are updated or any node performs its step.
     */
    public void preRound() {
        // No implementation here! Add your code to the CustomGlobal.java
        // file in your project.
    }

    /**
     * The framework calls this method after each round. I.e. after each node has
     * executed its step, and the interference has been checked, but before the
     * graph is redrawn (in GUI mode).
     */
    public void postRound() {
        // No implementation here! Add your code to the CustomGlobal.java
```

Árvore de diretórios

Sinalgo Projects

- Sinalgo disponibiliza 6 projetos
- Um projeto é uma pasta dentro do diretório 'src/projects/' do Sinalgo



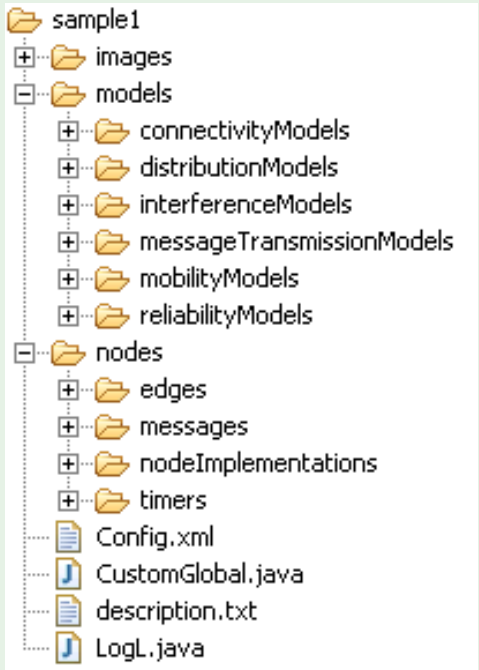
Árvore de diretórios

Sinalgo Projects

- Sinalgo disponibiliza 6 projetos
- Um projeto é uma pasta dentro do diretório 'src/projects/' do Sinalgo

Criando um novo projeto

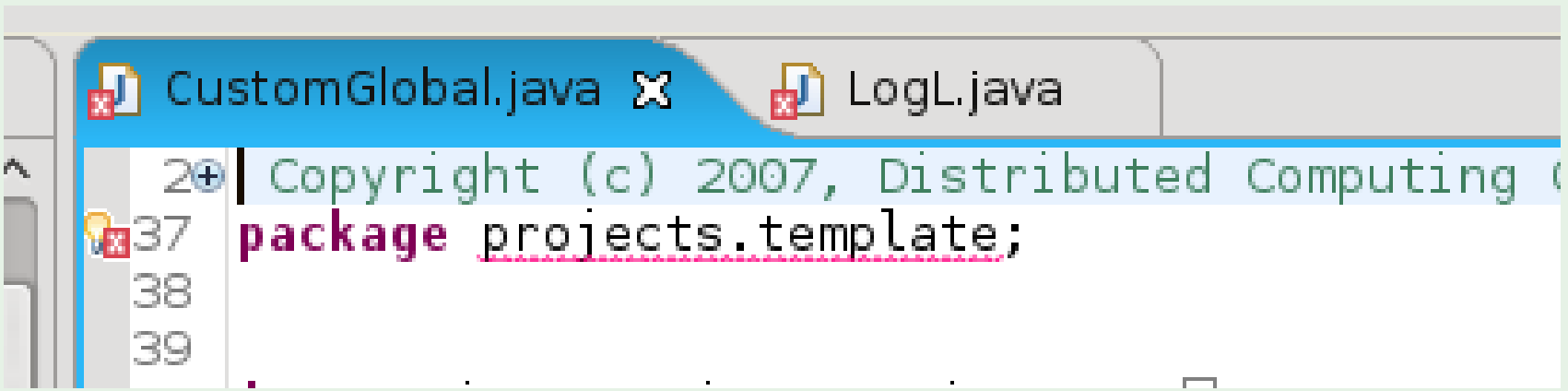
1. Faça uma copia da pasta 'src/projects/template/' para o mesmo diretório
2. Modifique o nome 'template (copy)' para o nome do seu projeto
3. Execute a aplicação novamente...
4. Pronto!



Erros

Erros aos copiar 'src/projects/template/'

É normal alguns erros de declaração dos packages.
Para corrigir basta modificar para:
package projects.nomeDoSeuProjeto¿



Dica!

Obs: É altamente recomendado gerar um projeto para cada algoritmo simulado. Contudo, isso freqüentemente resulta em código comum. Ao invés de copiar o mesmo código para todos os projetos, é preferível criar um novo projeto que armazenará esse código-comum e do qual todos os outros projetos terão acesso.

Meu Primeiro Projeto

Exemplo PingPong

Vamos criar uma simples simulação que demonstra os conceitos básicos.
Copie o diretório 'src/projects/template/' e modifique o nome para 'pingPong'

Meu Primeiro Projeto

Exemplo PingPong

Vamos criar uma simples simulação que demonstra os conceitos básicos.
Copie o diretório 'src/projects/template/' e modifique o nome para 'pingPong'

Comportamento PingPong

- Nesta simulação, dois nós vão trocar mensagens entre si.
- Os nós geram cores em RGB de modo aleatório.
- Cada cor gerada deve ser anexada em uma mensagem, que será enviada para o vizinho.
- Ao receber uma mensagem:
  - O nó deve alterar sua cor conforme os valores RBG recebidos.
  - Gerar uma nova cor RGB e enviar por broadcast.

Meu Primeiro Projeto

Por onde começo a implementar?

R: Em todos projetos temos que implementar:

1. Comportamento do nó
2. Modelos
3. Criar um arquivo de configuração

Como implementar o comportamento do nó?

Básica implementação

1. Cada nó simulado é uma instância (sub-classe) da classe **"sinalgo.nodes.Node"**.
2. Salve a classe em **'nodes/nodeImplementations/'** do projeto pingPong.
3. Cada nó implementa seu próprio comportamento:
   - Ex: cada nó tem um método que é chamado quando uma mensagem é recebida. Também existe um método para enviar mensagens para seus vizinhos.
   - Você deve implementar o comportamento adequado para cada mensagem recebida.
4. Cada nó tem sua própria instância de Mobilidade, Confiabilidade, Interferência, e Conectividade.
5. Existe um modelo de Transmissão de Mensagens que é global.

Como implementar o comportamento do nó?

Criando um nó

1. Crie uma classe que estenda **'sinalgo.nodes.Node'** chamada "PingPongNode.java'.
2. handleMessages() é o método onde o comportamento deve ser implementado.

OBS

Obrigatoriamente, deve-se implementar o método handleMessages() e, opcionalmente, qualquer um dos métodos abstratos da classe sinalgo.nodes.Node

```
1  package projects.pingPong.nodes.nodeImplementations;
   import ...
3
   public class pingPongNode extends Node {
5    private int local_r, local_g, local_b;
     @Override
7    public void handleMessages(Inbox inbox) {}

9    @Override
     public void preStep() {}
11
     @Override
13   public void init() {}

15   @Override
     public void postStep() {}
17
     @Override
19   public void checkRequirements() throws
       WrongConfigurationException {}
   }
```

Listing 1: "pingPongNode.java"

Como implementar o comportamento do nó?

Criando uma mensagem

1. Antes de implementar o handleMessages(), vamos criar nossa **Mensagem**.
2. A comunicação no Sinalgo é feita através de mensagens.
3. Toda mensagem deve herdar da classe **'sinalgo.nodes.messages.Message'**
4. As mensagens devem ser salvas no diretório:
   - **'nodes/messages/'** do projeto pingPong.
5. A classe abstrata Message requer que seja implementada apenas um método.
   - **public Message clone( );**

Como implementar o comportamento do nó?

OBS

Quando um nó envia uma mensagem para um nó vizinho, assume-se que o destino recebe o conteúdo da mensagem que foi enviado pelo método send( ). O framework, contudo, não tem como saber se o transmissor ainda tem uma referência para o objeto de mensagem enviado, e conseqüentemente, ele pode alterar a mensagem enviada. Para enviar tais problemas, o framework envia cópias separadas para todos os nós que recebem uma mensagem.

```
package projects.pingPong.nodes.messages;

import sinalgo.nodes.messages.Message;

public class PingPongMessage extends Message {

  private int r, g, b;

  public PingPongMessage(int i, int j, int k) { this.r = i;
    this.g = j; this.b = k; }

  @Override
  public Message clone() { return new PingPongMessage(this.r
    , this.g, this.b); }

  /*Getters and Setters*/
}
```

Listing 2: "PingPongMessage.java"

Como implementar o comportamento do nó?

Criando uma mensagem

- A classe é auto-explicativa.
- Basicamente definimos os campos da mensagem e a implementação do método clone( ), que será usado para gerar uma cópia da mensagem.

Como implementar o comportamento do nó?

Voltando a implementação do nó...

1. O método mais importante é o handleMessage(Inbox inbox).
2. handleMessages(Inbox inbox) é o método onde o comportamento deve ser implementado.
3. Esse método é usado para tratar as mensagens que o nó recebeu.
4. Cada nó armazena as mensagens que recebe em uma instância da classe Inbox.
5. Inbox provê um iterador para acessar cada mensagem armazenada.
6. Para cada mensagem, o interador armazena meta-informações, como o transmissor (sender) da mensagem.

```
1  public void handleMessages(Inbox inbox)
     while (inbox.hasNext())
3      Message message = inbox.next();
       if(message instanceof PingPongMessage)
5        PingPongMessage pkt = (PingPongMessage) message;
         this.local_r = pkt.getR();
7        this.local_g = pkt.getG();
         this.local_b = pkt.getB();
9
         this.setColor(new Color(local_r, local_g, local_b));
11
         UniformDistribution ud = new UniformDistribution(0,
       255);
13
         pkt.setR( (int) ud.nextSample());
15       pkt.setG( (int) ud.nextSample());
         pkt.setB( (int) ud.nextSample());
17
         PingPongTimer timer = new PingPongTimer(pkt);
19       timer.startRelative(1, this);
```

Listing 3: "handleMessage"

Como implementar o comportamento do nó?

Voltando a implementação do nó...

1. A classe pingPongNode.java já é capaz de receber e tratar as mensagens recebidas.
2. **Mas como mandar mensagens?**
3. O que é **PingPongTimer?**

Como implementar o comportamento do nó?

Temporizadores

1. Temporizadores servem para especificarmos quando uma determinada ação deverá ser executada.
2. No nosso exemplo, criamos um temporizador com a seguinte declaração:
   - **PingPongTimer timer = new PingPongTimer(pkt);**
   - Perceba que passamos como parâmetro, para o construtor, a mensagem que queremos transmitir. Mais a frente, explicaremos o porque.

Como implementar o comportamento do nó?

Considerações

- Iniciamos o temporizador através do seguinte comando:
  - **timer.startRelative(1, this);**
  - O primeiro parâmetro indica em qual turno o temporizador será disparado. O segundo parâmetro indica qual objeto está disparando o temporizador.
  - O tempo 1 indica que o temporizador será disparado no próximo round.

Como implementar o comportamento do nó?

Criando um temporizadores

1. Todo temporizador herda da classe **'sinalgo.nodes.timers.Timer'**
2. Crie uma classe chamada PingPongTimer.java dentro de 'nodes/timers/':
3. Basicamente, implementamos o método **fire( )**, que é a ação a ser executada quando o temporizador for disparado.
4. No nosso exemplo, queremos que:
   - Toda vez que o temporizador for disparado, ou seja, quando fire() é executado,
   - o método broadcast do nó que iniciou o temporizador será disparado para enviar a mensagem.

```
1  package projects.pingPong.nodes.timers;

3  import ...

5  public class PingPongTimer extends Timer {
     private PingPongMessage pkt = null;
7
     public PingPongTimer(PingPongMessage pkt) {
9      this.pkt = pkt;
     }
11
     @Override
13   public void fire() {
       ((pingPongNode)node).broadcast(this.pkt);
15   }
   }
```

Listing 4: "PingPongTimer.java"

Como implementar o comportamento do nó?

Criando um método para iniciar a comunicação

1. Como iniciar a troca de mensagens?
2. Adicione o seguinte método na classe PingPongNode.java:

Como implementar o comportamento do nó?

Criando um método para iniciar a comunicação

1. Como iniciar a troca de mensagens?
2. Adicione o seguinte método na classe PingPongNode.java:

```
@NodePopupMethod(menuText = "Init pingPong")
public void initPingPong() {
  PingPongMessage msg = new PingPongMessage(0,0,0);
  PingPongTimer t = new PingPongTimer(msg);
  t.startRelative(1, this);
}
```

Listing 6: "NodePopupMethod"

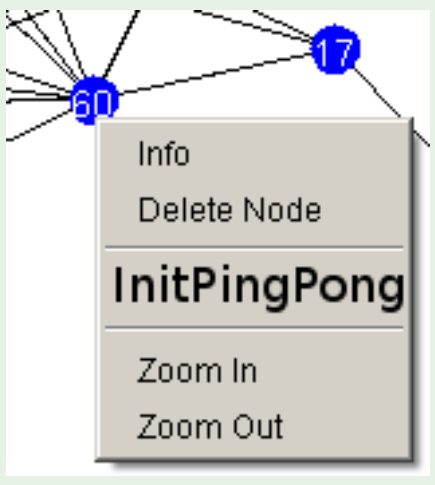
Como implementar o comportamento do nó?

Criando um método para iniciar a comunicação

1. Como iniciar a troca de mensagens?
2. Adicione o seguinte método na classe PingPongNode.java:

```
@NodePopupMethod(menuText = "Init pingPong")
public void initPingPong() {
  PingPongMessage msg = new PingPongMessage(0,0,0);
  PingPongTimer t = new PingPongTimer(msg);
  t.startRelative(1, this);
}
```

Listing 7: "NodePopupMethod"



Como implementar o comportamento do nó?

Arquivo de configuração

1. Nossa simulação já está pronta!!!
2. Falta somente configurar o projeto:
   - Edite o arquivo **pingPong/config.xml**

```
<Document>
  <Framework>
    ...
  </Framework>
  <Custom>
    ...
  </Custom>
</Document>
```

Listing 8: "Config.xml"

```
<Document>
  <Framework>
    ...
  </Framework>
<Custom>
  <QUDG rMin="30" rMax="50" ProbabilityType="constant"
    connectionProbability="0.6"/>

  <SINR alpha="2" beta="0.7" noise="0"/>

  <RandomWayPoint>
    <Speed distribution="Gaussian" mean="10" variance="20" /
    >
    <WaitingTime distribution="Poisson" lambda="10" />
  </RandomWayPoint>

  <RandomDirection>
    <NodeSpeed distribution="Gaussian" mean="10" variance="
    20" />
    <WaitingTime distribution="Poisson" lambda="10" />
    <MoveTime distribution="Uniform" min="5" max="20" />
  </RandomDirection>
</Custom>
</Document>
```

Listing 9: "Config.xml"

Git é vida!

git clone
git@github.com:BrunoPereiraSantos/aula-simuladores-redes-sem-fio.git

Tarefa 1

Exemplos do Sinalgo

1. Execute os 6 exemplos do Sinalgo.
2. Descreva a finalidade do exemplo.
3. Quais conceitos visto em sala de aula que são demonstrados em cada exemplo.
4. Descreva as limitações de cada exemplo.
5. Quais os pontos fortes e fracos do Sinalgo?

Tarefa 2

Árvore para coleta de dados

1. Faça uma inundação para descobrir o menor caminho (em saltos) de cada nó para uma Estação Base (EB).
2. A EB deve ser iniciada através de **@NodePopupMethod**
3. As mensagens podem ser do tipo rota ou dados
4. O nó deve identificar se a mensagem é de construção de rota ou de dados.
   - Se a mensagem é de construção de rota
     - O nó deve atualizar sua rota para a EB.
   - Se a mensagem é de dados
     - O nó deve mandar uma mensagem unicast para o próximo salto até a EB, caso exista uma rota válida

Tarefa 2

Árvore para coleta de dados

1. A EB ao receber uma mensagem deve imprimir no Output informações sobre a mensagem coletada.
2. Cada nó após ter um caminho válido para EB, deve ser permitido enviar dados periodicamente (10 rounds) para EB, isto deve ser ativado através de um **@NodePopupMethod**.
3. O payload da mensagem de dado pode ser uma amostra de temperatura ou alguma variável de ambiente geralmente analisada por RSSF.
4. A mensagem não pode trafegar mais que um tempo de vida estabelecido (Ex: 30 saltos)

Tarefa 2

DICAS

1. Use Tools.java para:
   - Escrever no output
   - Manipular erros
   - Informações sobre a simulação
   - Acessar conjunto de nós
   - Acessar mensagens
   - Usar métodos relacionados com a GUI
   - Parar a simulação
2. Como mandar mensagens unicast?
   - R: documentação sinalgo! =D

Tarefa 2

DICAS

1. Quais classes devo criar?
   - Uma classe para representar o nó (Ex: "MyNode.java")
     - A classe deve conter uma referência para o próximo salto até a EB.
     - Deve ter uma variável para indicar sua distância até a EB.
     - Criar um método menu popup para iniciar a EB na classe "MyNode.java".
     - Implemente um mecanismo para evitar loop das mensagens (Dica: use o campo número de saltos da mensagem)

Tarefa 2

DICAS

1. Quais classes devo criar?
   - Uma ou duas classes para representar as mensagens (rota e dados)
   - Cabeçalho indicado:
     - número de seqüência.
     - tempo de vida.
     - destino e origem.
     - nó que encaminhou a mensagem
     - número de saltos até a EB
     - tipo (se fez uma classe só)

Tarefa 2

DICAS

1. Quais classes devo criar?
   - Um ou dois timers para enviar as mensagens do flood (para descoberta das rotas) e para enviar periodicamente as mensagens de dados.
   - Implementar um temporizador que armazene um pacote.
   - implementar adequadamente o método **fire()**.

Até a próxima aula!