JAVA RMI

Instruções gerais:

- Salve cada classe/interface em um arquivo .java separado
- Os exercícios são incrementais, não é necessário criar diretórios separados para cada um deles
- Utilize o mesmo diretório base para os arquivos do cliente e do servidor:

```
~/ap2/cliente
~/ap2/servidor
 ou apenas
~/ap2
```

- Não esqueça de rodar rmiregistry a partir da pasta raiz das classes (compiladas)

```
cd ~/ap2
rmiregistry
```

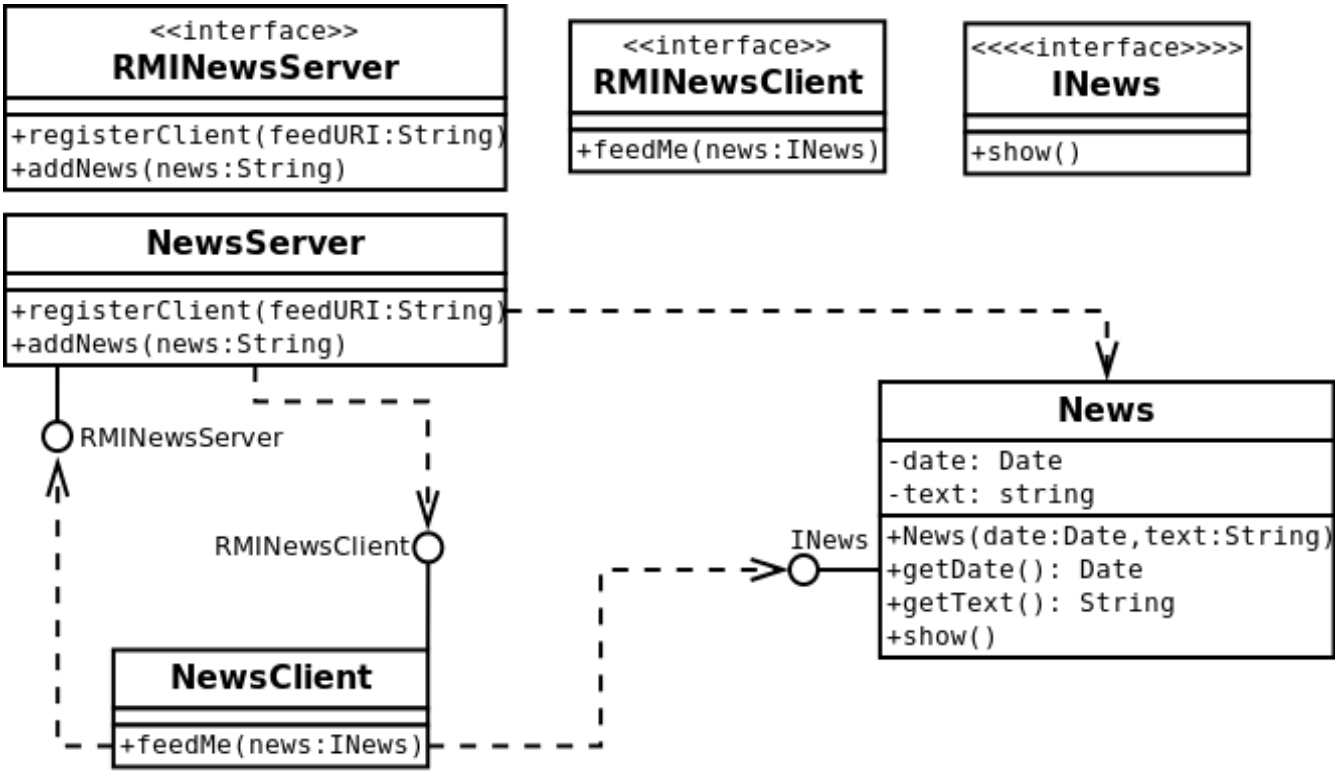
O objetivo do exercícios é construir um sistema de distribuição de notícias (mensagens) com o seguinte funcionamento:

- O servidor inicia
- Alguns clientes iniciam
- O cliente informa ao servidor uma URI onde escuta por notícias (`registerClient`)
- Os clientes enviam notícias (`addNews`)
- O servidor envia a notícia para todos os clientes (`addNews`)

Observações:

- Todas as chamadas cliente <-> servidor devem ser através de RMI
- Não importa a ordem em que forem entregues as notícias
- O cliente só recebe as notícias enviadas após sua conexão

Este sistema é representado na figura a seguir:



1 - Considere a classe News:

```
import java.util.Date;

public class News implements INews {
    private Date date;
    private String text;

    News(Date date, String text) {
        this.date = date;
        this.text = text;
    }

    public Date getDate() { return date; }

    public String getText() { return text; }

    public void show() {
        System.out.println("<" + date.toString() + "> " + text);
    }
}
```

E sua interface:

```
public interface INews {
    public void show();
}
```

a) Altere a classe News para que possa ser passada como parâmetro em uma chamada RMI

- Dica: Deve implementar...

2 – Considere a interface RMINewsServer e a classe NewsServer (o lado servidor)

```
public interface RMINewsServer {
    public void registerClient(String feedURI) /* throws... */;
    public void addNews(String news);
}

public class NewsServer  implements RMINewsServer {
  public NewsServer() /* throws... */ {}

  public void registerClient(String feedURI) {
      System.out.println("request from "+feedURI);
      //conecta ao cliente
      //adiciona objeto remoto na lista de clientes
  }

  public void addNews(String news) {
      News n = new News(new Date(), news);
      //envia n para todos os clientes
  }
}
```

a) Altere a interface `RMINewsServer` para que possa ser um objeto de invocação remota
Dica: deve implementar...
b) Altere a implementação `NewsServer` para que possa ser instanciado como objeto de invocação remota
Dica: precisa estender...
c) Crie um programa que instancie a classe `NewsServer` e registre no rmiregistry com o nome "NewsServer"
Não esqueça de executar rmiregistry

3 – Considere a interface e a classe cliente:

```
import INews;

public interface RMINewsClient /* impl... */ {
    public void feedMe(INews news) /*throws ...*/;
}

public class NewsClient /* extends... */ implements RMINewsClient {
    public NewsClient() /* throws... */ {}

    @Override
    public void feedMe(INews news) {
        news.show();
    }
}
```

a) Altere a classe e a interface para serem usadas em RMI
b) Crie um programa que instancie e registre a `NewsClient`
Dica: Como são diversos clientes, registre como "NewsClient" + <inteiro aleatório>

4 – Implemente:

a) Implemente os métodos `registerClient` e `addNews` da `NewsServer`
Lembrete: os métodos são chamados concorrentemente pelos clientes
b) Altere o programa do exercício 3 para que registre-se utilizando `registerClient` e adicione algumas noticias utilizando `addNews`
Dica: Adicione alguns sleep entre as chamadas de addNews