Java Passing Message Interface JPMI

Dr. Mario Rossainz López

Programación Concurrente y Paralela

Otoño de 2022

NRC: 60898

Programación Concurrente con Paso de Mensajes en Java (JPMI)

- Un proceso se crea implementando la interface: ***Jpmi.Proceso*** la cual especifica un método abstracto: ***public void run();*** y es similar a la interface de java: ***java.lang.Runnable***.

```
public interface Jpmi.Proceso
   {
    public void run();
   }
```

Creando Procesos en JPMI

- Un proceso es una clase en Java que implementa la interface ***Jpmi.Proceso*** y debe proporcionar una implementación al método ***run( )***. Éste método contendrá el código que el proceso deberá ejecutar cuando éste método ***run( )*** sea invocado por otro proceso.

Creando Procesos en JPMI

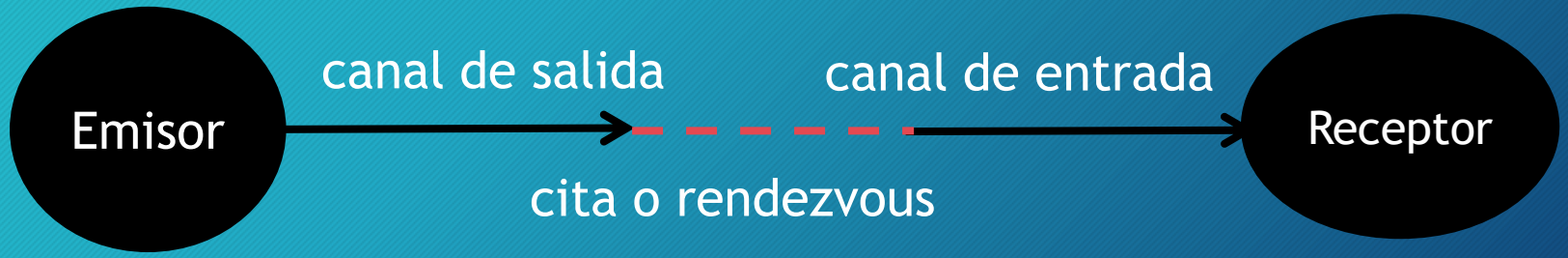
```
public class Mi_proceso implements Jpmi.Proceso
{
    //variables de instancia locales de Mi_proceso

    public Mi_proceso(canales y parámetros)
        {
            // Constructor
        }
    public void run()
       {
        // hacer algo. . .
        }
}
```

Uso de canales

- El canal es el medio de comunicación más simple entre dos procesos en la programación concurrente con paso de mensajes. Un canal es un objeto pasivo que sirve de intermediario entre un proceso que envía un mensaje y un proceso que lo recibe y es en donde toma lugar la sincronización y las políticas de planificación y comunicación de los procesos que lo utilizan. En otras palabras, un canal es un “monitor” que controla la comunicación de dos procesos.

Uso de canales

- Un canal sincroniza a los procesos (synchronized) que estan conectados a él acorde al principio de ***cita (rendezvous)*** . El proceso que envía el mensaje es bloqueado (wait) por el canal y espera para continuar su ejecución hasta que el proceso que recibe el mensaje esta listo para ello (notify). De igual manera el proceso que recibe el mensaje es bloqueado (wait) por el canal y espera para continuar su ejecución hasta que el proceso que envía el mensaje este listo para hacerlo (notify).



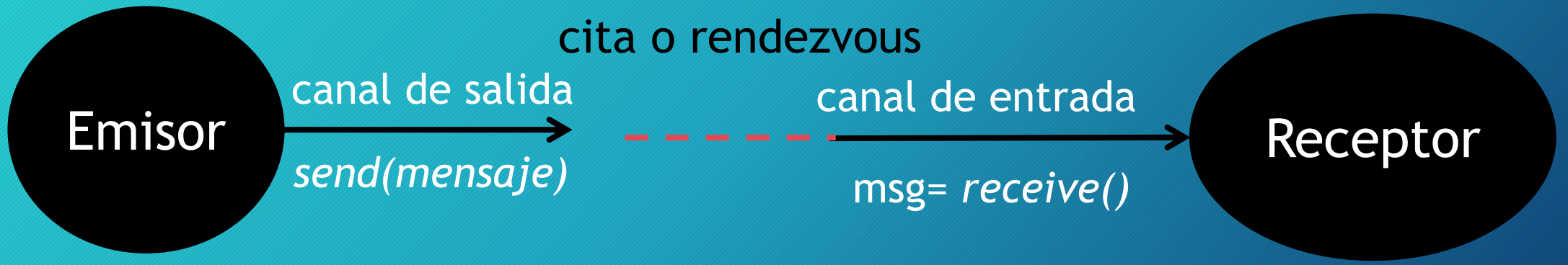
Uso de canales en JPMI

- En JPMI los canales se crean haciendo uso de la clase ***Jpmi.CanalSimple***, la cual proporciona un método para enviar un mensaje por parte de un proceso emisor: ***send(. .)*** y un método para recibir un mensaje por parte de un proceso receptor ***receive(***. . .).
- Dicho de otra forma: Un proceso que quiere enviar un mensaje a otro proceso deberá utilizar un ***"canal de salida"*** para utilizar la operación de envío ***send(***. . .***)***; mientras que un proceso que quiere recibir un mensaje de otro proceso deberá utilizar un ***"canal de entrada"*** para utilizar la operación de recepción ***receive(***. . .***)***.

Uso de canales en JPMI

```
public class Jpmi.CanalSimple
{
  // variables de instancia locales

  public CanalSimple() { //constructor 1 }

  public CanalSimple(boolean . . .){ //constructor 2 }

  public void send(Object mensaje) { . . . }

  public Object receive() { . . . }
}
```

Uso de canales en JPMI



EJEMPLO: Emisor/Receptor en JPMI

```
public class Mensaje
{
 public String id;
 public int dato;

 public Mensaje()
   {
    id="";
    dato=0;
   }

 public Mensaje(String id,int dato)
   {
    this.id=id;
    this.dato=dato;
   }
}
```

El proceso Emisor en JPMI

```
import Jpmi.*;

public class Emisor implements Proceso
{
 CanalSimple canal[];
 Mensaje msg;

 public Emisor(Mensaje msg,CanalSimple... canal)
  {
   this.msg=msg;
   this.canal=canal;
  }
```

```
 public void run()
  {
   for(int i=0;i<canal.length;i++)
   {
    System.out.println("Emisor "+msg.id+
                       " enviando “+msg.dato
                       +" al receptor...");
    canal[i].send(msg);
   }
  }
}
```

El proceso Receptor en JPMI

```
import Jpmi.*;

public class Receptor implements Proceso
  {
   CanalSimple canal[];
   Mensaje msg;
   String name;

   public Receptor(String name,CanalSimple... canal)
   {
    this.name=name;
    this.canal=canal;
   }
```

```
public void run()
  {
   for(int i=0;i<canal.length;i++)
   {
     msg=(Mensaje)canal[i].receive();
     System.out.println("RECEPTOR "+name+
                        " recibiendo "+
                        msg.dato+
                  " del emisor "+msg.id);
   }
  }
}
```

Composiciónes de Procesos en JPMI

- En JPMI un proceso inicia su ejecución cuando su método ***run( )*** es invocado. Esta invocación se lleva a cabo dentro de lo que se conoce como una "COMPOSICIÓN de PROCESOS" la cual especifica la manera en la que los procesos que se comunican mediante canales pueden ser lanzados a ejecución: en secuencia, en paralelo o mediante una elección.

- Dicho de otra forma, los procesos se pueden ejecutar en 3 tipos de composiciones: Secuencial, Paralelo o Alternativa.

- Una composición de procesos es en sí misma un proceso. Esto significa que puede haber anidación de composiciones, es decir, que dentro de una composición secuencial por ejemplo, puede haber una o más composiciones secuenciales, paralelas o alternativas.

- Una composición invoca automáticamente los métodos ***run( )*** de sus subprocesos.
- Se puede construir entonces toda una red de procesos conectados entre sí mediante canales y lanzados a ejecución mediante composiciones para resolver problemas simples o complejos.

Composición Secuencial de Procesos en JPMI

```
public class  Jpmi.Secuencial implements Jpmi.Proceso
  {
    //variables de instancia locales

    public Secuencial(Jpmi.Proceso p[ ])
     {
       //Constructor de la composición secuencial
       //que recibe el conjunto de procesos p[ ] e ejecutar en secuencia...
     }

    public void run()
      {
       //ejecución secuencial del conjunto de procesos p[ ]
      }
  }
```

Composición Paralela de Procesos en JPMI

```
public class  Jpmi.Paralelo implements Jpmi.Proceso
  {
    //variables de instancia locales

    public Paralelo(Jpmi.Proceso p[ ])
     {
       //Constructor de la composición paralela
       //que recibe el conjunto de procesos p[ ] e ejecutar en paralelo...
     }

    public void run()
      {
       //ejecución simultanea (en paralelo) del conjunto de procesos p[ ]
      }
  }
```

Composición Alternativa de Procesos en JPMI

```
public class  Jpmi.Alternativa implements Jpmi.Proceso
  {
    //variables de instancia locales

    public Alternativa(Jpmi.Guarda g[ ])
     {
       //Constructor de la composición alternativa que recibe un conjunto objetos “Guarda” asociados a
       //los procesosque habrán de ejecutarse en elección (choice)
     }

    public int select()
      {
        //Método que se utiliza para elegir el índice del proceso a ejecutar dentro del arreglo de guardas
      }

    public void run()
      {
       //ejecución selectiva del conjunto de procesos asociados al conjunto de guardas...
      }
  }
```

Guardas de procesos en JPMI

```
public class  Jpmi.Guarda
  {
    //variables de instancia locales

    public Guarda(boolean activo, Jpmi.Proceso p)
     {
       //Constructor1 de la clase guarda
       //que recibe un proceso y lo define como proceso activo
     }

    public Guarda(Jpmi.CanalSimple canal)
      {
       //Constructor2 que define un Guarda asociándole un canal...
      }
  }
```

Programa Principal: Emisor/Receptor

```
import Jpmi.*;
public class EmisorReceptorPar
  {
   public static void main(String args[])
     {
      CanalSimple c1=new CanalSimple();

      Paralelo par=new Paralelo(new Proceso[]{
                                  new Emisor(new Mensaje("A",100),c1),
                                  new Receptor("1",c1),
                                 });


      par.run();
     }
  }
```