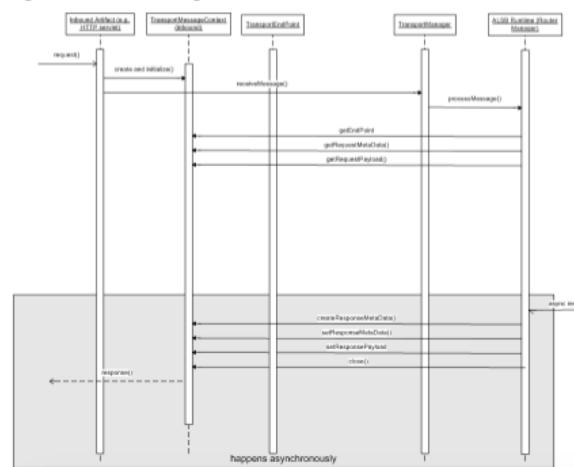


# Diagrama de Secuencia

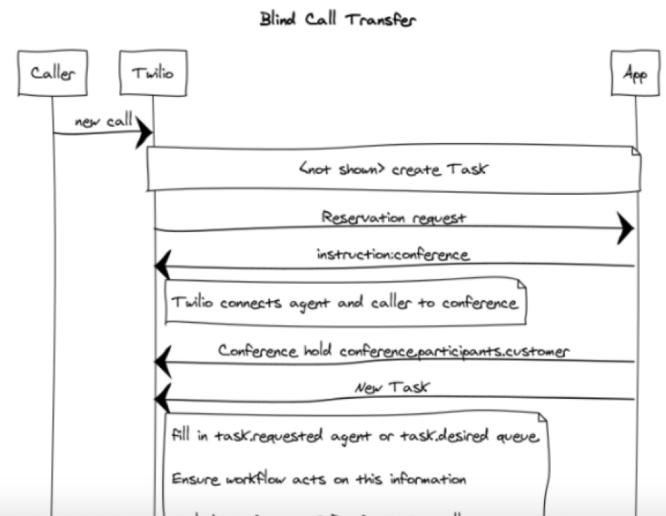
Dr. Santiago Vidal

ISISTAN-CONICET

## UML Sequence Diagrams



## Blind Transfer



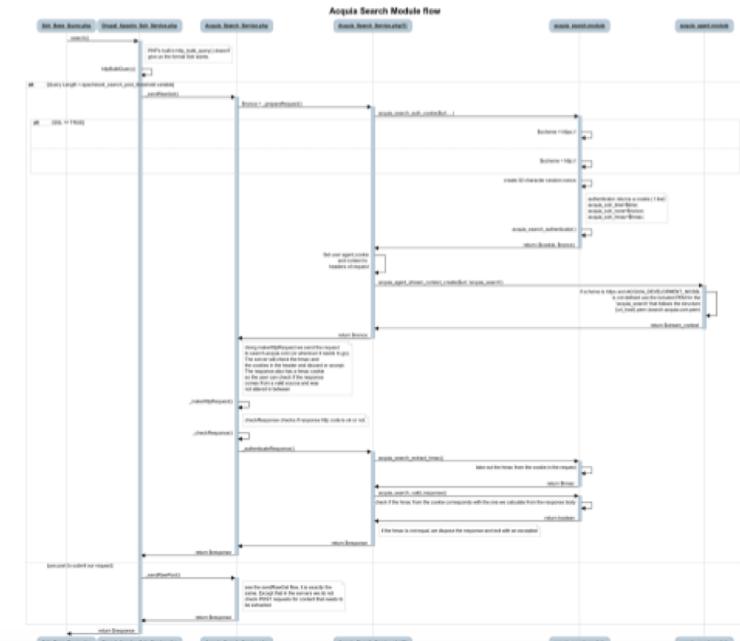
Drupal™

time for the 2020 Drupal project survey. Now is your opportunity to influence the direction of Drupal.

Submit your Feedback

## sequence diagram

## Sequence diagram / flow diagram

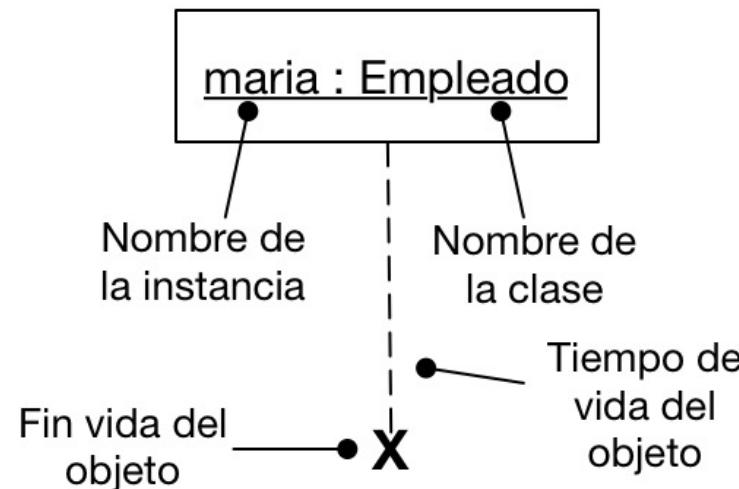


# Concepto

- Se usan para modelar el flujo de control de una operación
- Muestran los mensajes intercambiados entre un conjunto de objetos para realizar una tarea específica
- Hace énfasis en el orden en que se envían los mensajes
- Tiene 2 componentes principales:
  - Objetos
  - Mensajes

# Objetos

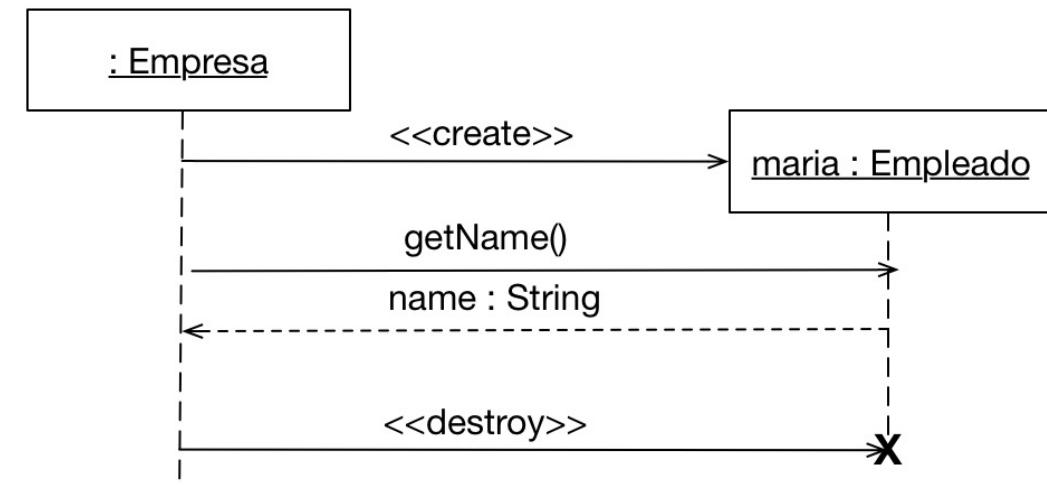
- Son instancias de las clases contenidas en un diagrama de clases
- Gráficamente el objeto se dibuja como un rectángulo con su nombre de instancia y clase subrayados



- La línea de vida indica el tiempo durante el que existe el objeto

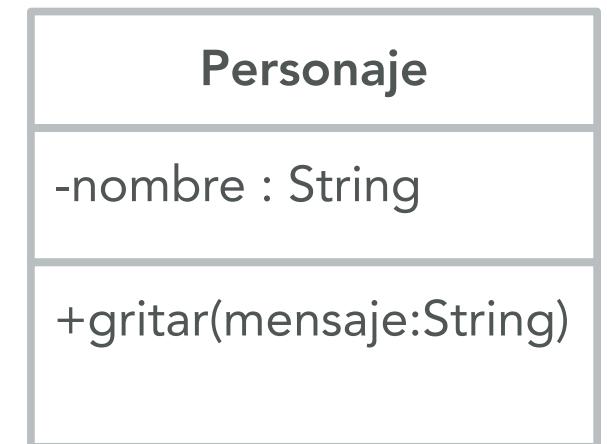
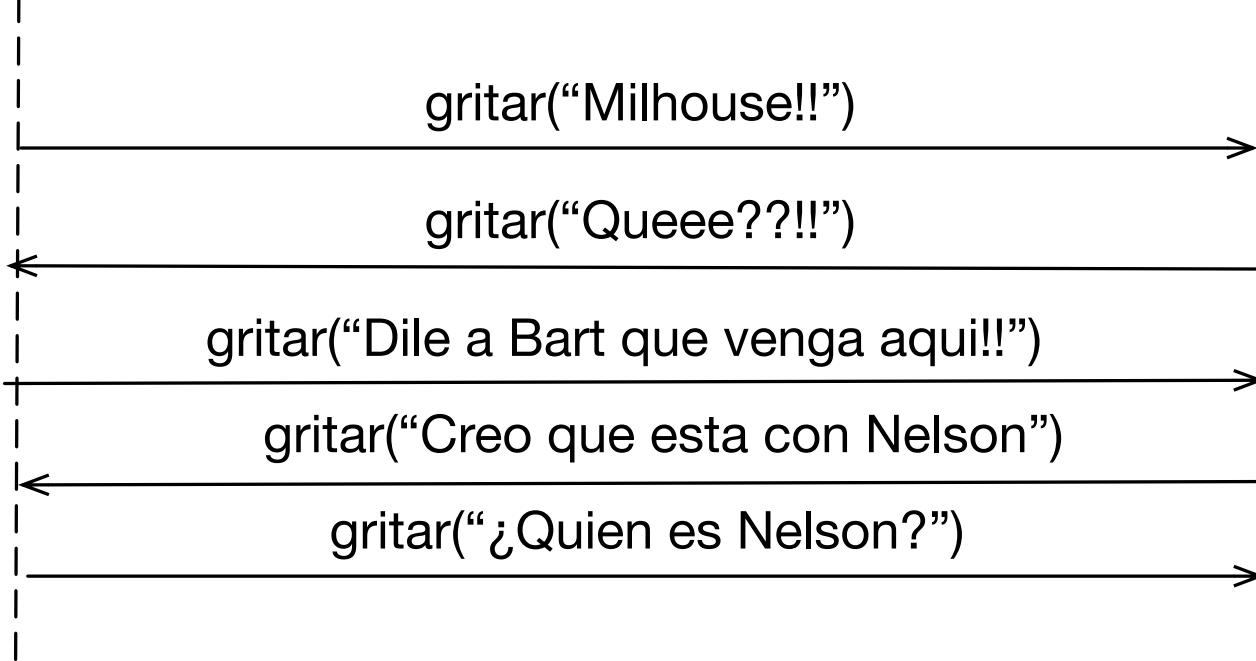
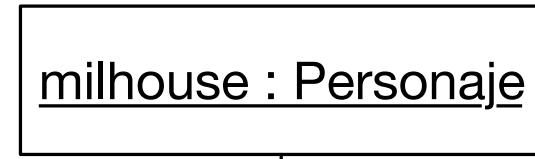
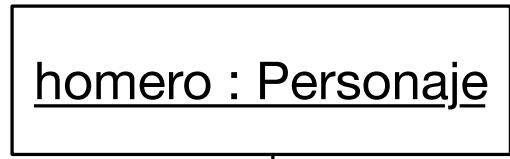
# Mensajes

- Los mensajes son la especificación de la comunicación entre objetos
- Permiten modelar distintos tipos de acciones
  - Call: invocación de una operación de un objeto
  - Return: valor de retorno de una operación
  - Create: creación de un objeto
  - Destroy: destrucción de un objeto



# Identificando llamados





# Diagrama desde código fuente

- Hagamos el diagrama de secuencia que inicia cuando un objeto **p** de la clase **Puerta** recibe el mensaje **validarEntrada** con un String como parámetro.

```
public class Puerta {  
  
    public void validarEntrada(Object o) {  
        String dni = (String) o;  
        Empleado e = this.obtenerDatos(dni);  
        boolean tienePermiso = this.validarDNI(dni);  
        if(tienePermiso) {  
            control.abrir(this);  
        }  
        else {  
            this.solicitarPermTemporal(e);  
            control.abrir(this);  
        }  
    }  
  
    public void solicitarPermTemporal(Empleado e) {  
        PermisoTemporal permiso = new PermisoTemporal();  
        permiso.set(this);  
        permiso.setDesde(Date.HOY);  
        permiso.setHasta(Date.MAÑANA);  
        permiso.setDescripcion("");  
        e.agregar(permiso);  
    }  
}
```

```

public class Puerta {

    public void validarEntrada(Object o) {
        String dni = (String) o;
        Empleado e = this.obtenerDatos(dni);
        boolean tienePermiso = this.validarDNI(dni);
        if(tienePermiso) {
            control.abrir(this);
        }
        else {
            this.solicitarPermTemporal(e);
            control.abrir(this);
        }
    }
}

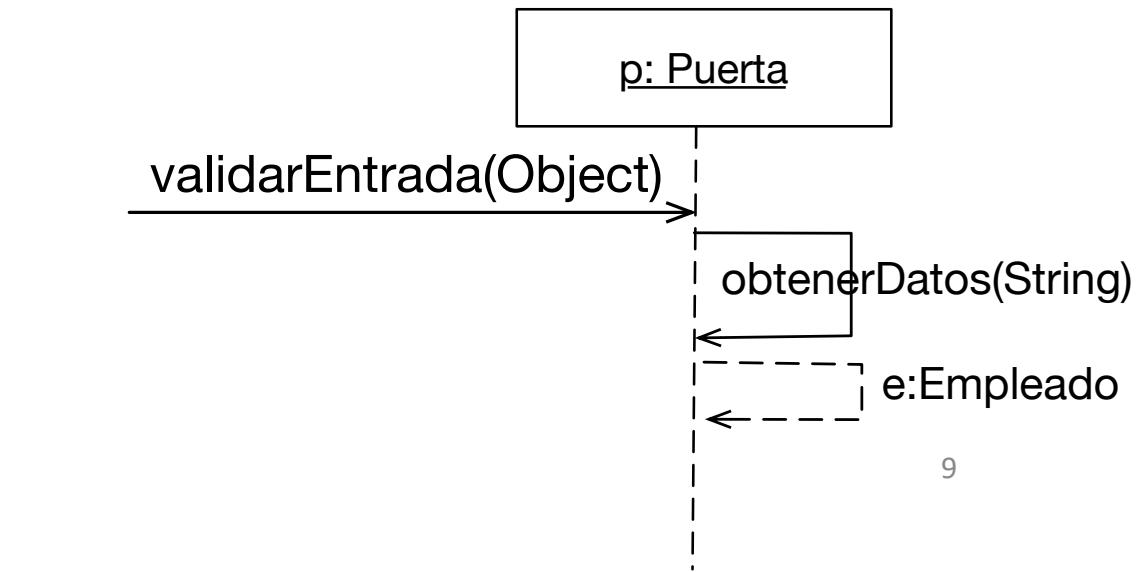
```

```

public void solicitarPermTemporal(Empleado e) {
    PermisoTemporal permiso = new PermisoTemporal();
    permiso.set(this);
    permiso.setDesde(Date.HOY);
    permiso.setHasta(Date.MAÑANA);
    permiso.setDescripcion("");
    e.agregar(permiso);
}

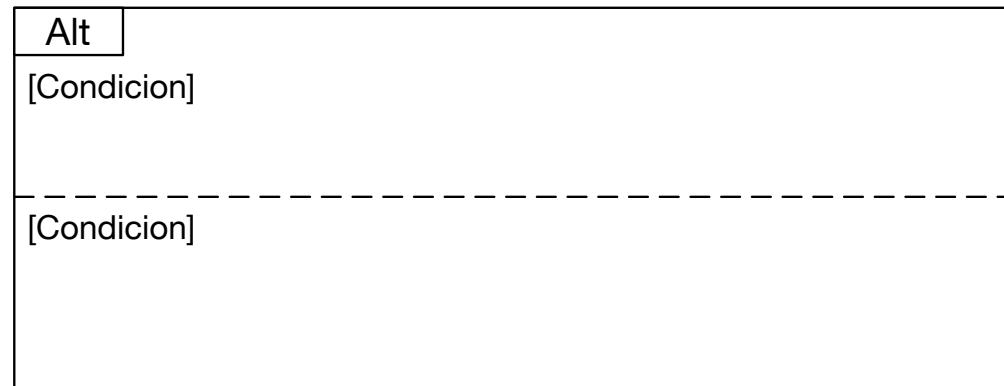
```

**¿cómo modelamos el if/else?**



# Frames

- Se usan para representar if/else, while, for...
- Operadores
  - Alt: es para múltiples fragmentos. Solo ejecuta el de la condición verdadera
  - Opt: un único fragmento que solo ejecuta cuando la condición es verdadera
  - Loop: es un fragmento que se ejecuta múltiples veces



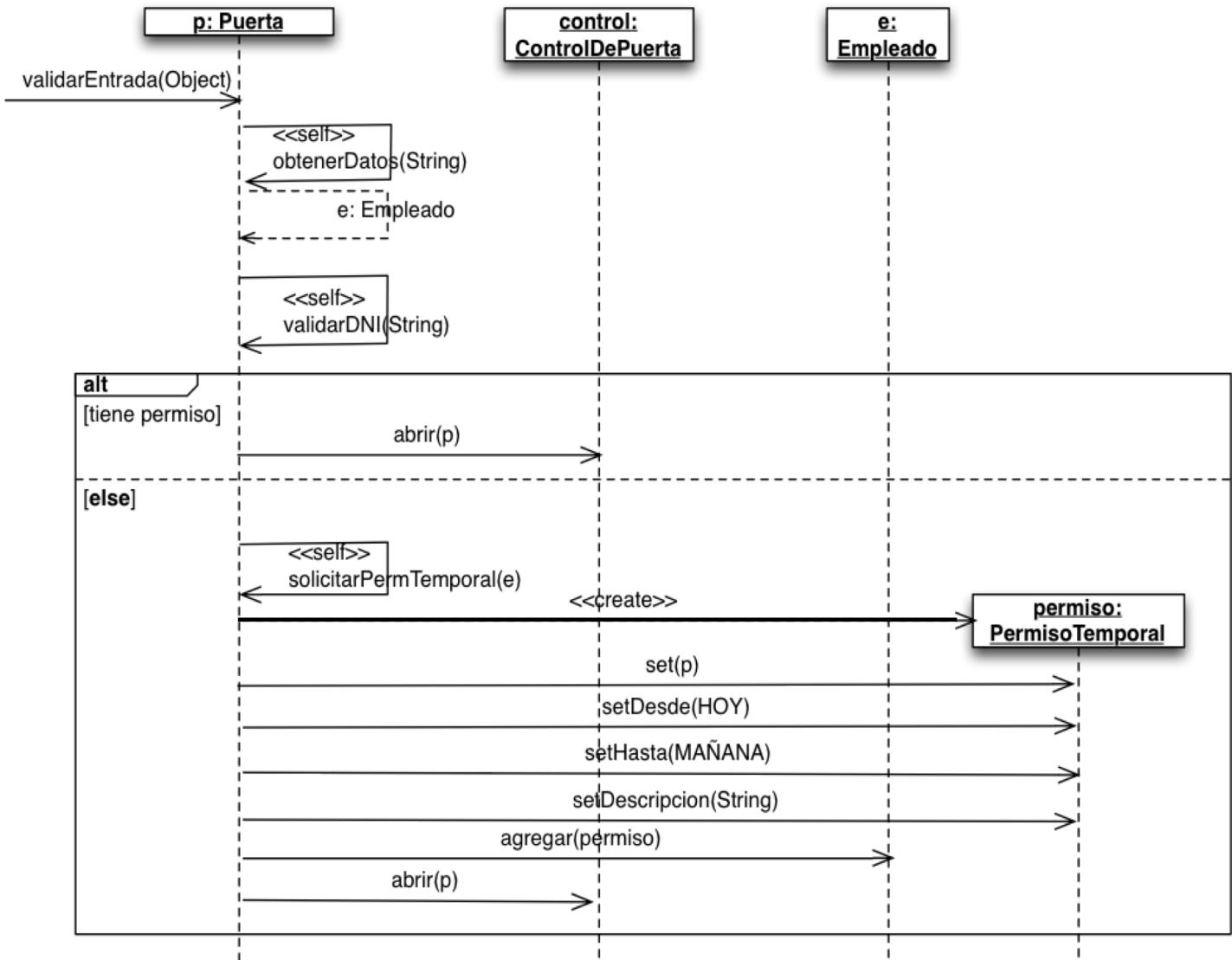
```

public class Puerta {

    public void validarEntrada(Object o) {
        String dni = (String) o;
        Empleado e = this.obtenerDatos(dni);
        boolean tienePermiso = this.validarDNI(dni);
        if(tienePermiso) {
            control.abrir(this);
        } else {
            this.solicitarPermTemporal(e);
            control.abrir(this);
        }
    }

    public void solicitarPermTemporal(Empleado e) {
        PermisoTemporal permiso = new PermisoTemporal();
        permiso.set(this);
        permiso.setDesde(Date.HOY);
        permiso.setHasta(Date.MAÑANA);
        permiso.setDescripcion("");
        e.agregar(permiso);
    }
}

```



# Diagrama desde enunciado

- Considere el siguiente relato de ficción entre diferentes personajes de la serie Game of Thrones® y modele el diagrama de secuencias correspondiente.
- Luego de fallecer el rey vigente de Westeros, **Robert Baratheon**, se desarrollan una serie de eventos entre los herederos al trono, sus hermanos **Stannis Baratheon** y **Renly Baratheon**.
- Inicialmente, el heredero **Stannis** recibe un mensaje *notificarMuerte* indicando el deceso de “robert”. Al enterarse de esta mala noticia, **Stannis** notifica a su hermano **Renly** enviando un mensaje con la misma firma y parámetros. Luego de recibir este último mensaje y, sabiendo que el primero en la línea de sucesión es **Stannis**, **Renly** envía un mensaje *proclamarseRey* a sí mismo proclamándose el verdadero rey de “Westeros”. A continuación, **Renly** envía un mensaje *informarProclamacion* a **Stannis**. Ante esta situación, **Stannis** le envía el mensaje *rendirse* a su hermano para evitar una confrontación militar. **Renly** puede contestar positiva o negativamente ante este pedido.
- Si decide rendirse, reconociendo a **Stannis** como el verdadero y único rey, entonces **Renly** recibe un mensaje *hacerPaces* de su hermano. A continuación, con el objetivo de reforzar sus fuerzas militares, **Stannis** le envía un mensaje *obtenerTropas* a su hermano para obtener las tropas que tenía a su cargo. Luego, el rey le envía el mensaje *reclutar* a cada uno de los soldados, indicando su identidad (la de **Stannis**) como parámetro (\*). Como consecuencia, cada soldado envía un mensaje *cambiarLealtad* a sí mismo para cambiar su lealtad al nuevo rey “**Stannis**”.
- Alternativamente, si **Renly** rechaza el pedido de rendición de su hermano, **Stannis** le envía un mensaje *asesinar a su bruja* religiosa **Melisandre** indicándole que asesine al usurpador del trono “**Renly**”. Con este fin, la bruja engendra un fantasma con forma de sombra y le indica que ataque al falso rey “**Renly**” a la “medianoche”. Bajo estas instrucciones, la sombra *destruye* a **Renly**. Una vez que el ataque finaliza, **Melisandre** termina con la existencia de este ente diabólico.
- (\*) Como no se sabe cuantos soldados están bajo el comando de **Renly**, el diagrama de secuencia debe tener en cuenta un número infinito de objetos. Recuerde que los *Iteradores* de Java pueden ser útiles para modelar este aspecto del enunciado. Un *Iterador* tiene dos métodos principales: *hasNext()* y *next()*. De esta forma, puede considerar que el mensaje *obtenerTropas* retorna un iterador a los soldados.