

**TSU Software, Materia: Programación, Clave 50086, Semestre 16-P**  
**Practica 11. Interfaces, Polimorfismo, Excepciones.**

## 1. Objetivos

Los objetivos buscados en esta práctica son los siguientes:

- Comprender y aplicar el concepto de interfaz a clases.
- Utilizado los conceptos de herencia e interfaz, comprender y utilizar el concepto de polimorfismos.
- Manejo básico de excepciones.

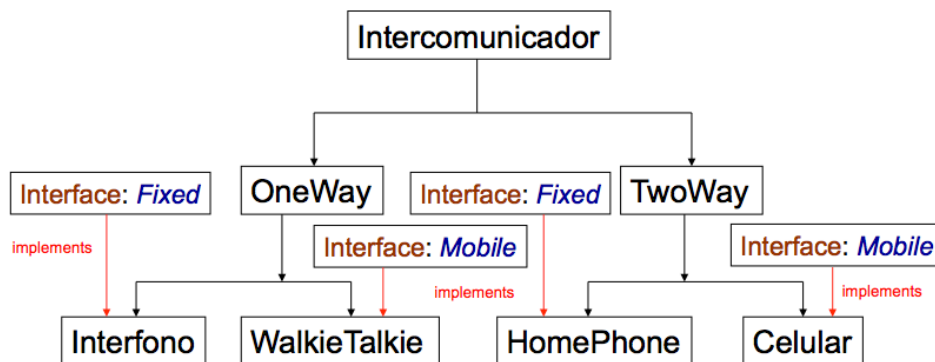
## 2. Ejemplo

En Java es posible definir una interfaz sin asociarla a ninguna clase y se le da un nombre. Se define cuales métodos y cuáles atributos (static final) la conforman. Al definir una clase se puede especificar cuál o cuáles interfaces serán parte de esa clase.

El polimorfismo (en la POO) es la propiedad que tiene un objeto de tipo A de mostrarse como de tipo B.

En el siguiente ejemplo analizaremos los dos tipos de relación:

- En cuanto a la herencia (*es un*)
- En cuanto a interfaz (*implementa*)



Intercomunicador: aparato que sirve para enviar y recibir mensajes

- ❑ Atributos:
  - Número de serie
- ❑ Métodos
  - Enviar mensaje
  - Recibir mensaje
  - toString

OneWay: Intercomunicador de una vía.

- ❑ Para enviar mensajes se debe oprimir el botón maestro
- ❑ Para escuchar mensajes se debe liberar el botón maestro
- ❑ Todos los aparatos escuchan los mensajes enviados

TwoWay: Intercomunicador de dos vías

- ☐ Se pueden enviar y recibir mensajes al mismo tiempo
- ☐ Se debe establecer una llamada desde un aparato a otro para poder iniciar la comunicación
- ☐ Los mensajes solo son escuchados por el aparato con el que se estableció la llamada

Fixed: aparato que se fija en algún lugar

- ☐ Ejemplo: cocina, baño, etc.

Mobile: aparato con encendido/apagado

- ☐ Cuando está apagado ninguna de sus funciones está disponible

Interfono: Intercomunicador OneWay Fijo

WallieTalkie: Intercomunicador OneWay Mobile

HomePhone: Intercomunicador TwoWay Fijo

Celular: Intercomunicador TwoWay Mobile

Ahora que se han explicado cada uno de los componentes del diagrama, empezaremos a declarar cada uno de los elementos, cuidando que las relaciones **es un** usemos la herencia **extends**, y la relación **implementa** usemos la implementación de interfaces **implements**.

Implementación de la superclase Intercomunicador, esta es la clase de mayor rango en la jerarquía.

```
public abstract class Intercomunicador {
    private String serialNum;
    public Intercomunicador( String n ){
        serialNum = new String( n );
    }
    protected abstract Boolean SendMessage( String m );
    protected abstract Boolean ReceiveMessage( Intercomunicador sender,
String m );
    public String toString(){
        return serialNum;
    }
    public String serialNum(){
        return serialNum;
    }
}
```

Implementación de las subclases OneWay y TwoWay que heredan de la superclase Intercomunicador.

```
import java.util.*;
public abstract class OneWay extends Intercomunicador {
    static final Integer LISTEN=0;
    static final Integer TALK=1;
    private Integer state;
    protected LinkedList<OneWay> listeners;
    Integer identity;
    public OneWay( String serialNum, Integer id ){
        super( serialNum );
        identity = id;
        state = LISTEN;
        listeners = new LinkedList<OneWay>();
    }
    public void Push(){
        state = TALK;
    }
    public void Release(){
        state = LISTEN;
    }
    protected Boolean AddListener( OneWay listener ){
        return listeners.add( listener );
    }
    protected void RemoveListener( OneWay x ){
        listeners.remove(x);
    }
    public Boolean SendMessage( String m ){
        Boolean success=true;
        for( Integer i=0; i<listeners.size(); i++ )
            success = success && listeners.get( i ).ReceiveMessage( this, m );
        return success;
    }
    protected Boolean ReceiveMessage( Intercomunicador sender, String m ){
        if( state != LISTEN ) return false;
        System.out.println(
            "\ns:" + sender.toString() +
            "\nr:" + this.toString() + "\nm:" + m );
        return true;
    }
    public String toString(){
        String [] s = {"LISTEN","TALK"};
        return super.toString()+" id:"+identity + " s:" + s[state];
    }
}

public interface Fixed {
    public void SetAddress( String addr );
}

public interface Mobile {
    public Boolean OnOff();
    public Boolean on();
}

public class Interfono extends OneWay implements Fixed {
    private String address;
```

```

    public Interfono( String serialNum, Integer id ){
        super( serialNum, id );
    }
    public void ConnectTo( Interfono x ){
        super.AddListener( x );
        x.AddListener( this );
    }
    public void SetAddress( String addr ) {
        address = addr;
    }
    public String toString(){
        return super.toString() + " A: " + address;
    }
}

public abstract class TwoWay extends Intercomunicador {
    public static final Integer IDLE=0;
    public static final Integer BUSY=1;
    public static final Integer RINGING=2;
    private Integer state;
    private String number;
    private Exchange ownExchange;
    private TwoWay other;
    public TwoWay( String serialNum, String num, Exchange e ){
        super( serialNum );
        state = IDLE;
        number = num;
        ownExchange = e;
        other = null;
        ownExchange.add( this );
    }
    public Boolean Dial( String num ){
        if( state != IDLE ) return false;
        other = ownExchange.Find( num );
        if( other == null ) return false;
        if( !other.Ring( this ) ) return false;
        state = BUSY;
        return true;
    }
    public void EndCall(){
        other.CallEnded();
        CallEnded();
    }
    public void CallEnded(){
        state = IDLE;
        other = null;
    }
}

public Boolean Ring( TwoWay sender ){
    if( sender == this ) return false;
    if( state != IDLE ) return false;
    state = RINGING;
    other = sender;
    return true;
}
    public void AnswerCall( String m ){
        if( state != RINGING ) return;
        state = BUSY;
    }
}

```

```

        SendMessage( m );
    }
    public String number(){
        return number;
    }

    public Boolean SendMessage( String m ){
        if( state != BUSY ) return false;
        if( other == null ) return false;
        return other.ReceiveMessage( this, m );
    }
    protected Boolean ReceiveMessage( Intercomunicador sender, String m ){
        if( state != BUSY ) return false;
        System.out.println(
            "\nr:" + this.toString() +
            "\ns:" + sender.toString() +
            "\nm:" + m );
        return true;
    }
    public String toString(){
        String [] s = {"IDLE","BUSY","RINGING"};
        return super.toString()+" #:"+number + " s:" + s[state];
    }
}

```

En el siguiente bloque de código veremos como implementar la herencia y la interfaz para los casos de estudio HomePhone y WalkieTalkie:

```

public class WalkieTalkie extends OneWay implements Mobile {
    private Boolean on;
    public WalkieTalkie( String serialNum, Integer id ){
        super( serialNum, id );
        on = false;
    }
    public void Push(){
        if( !on ) return;
        super.Push();
    }
    public void Release(){
        if( !on ) return;
        super.Release();
    }
    public Boolean SendMessage( String m ){
        if( !on ) return false;
        return super.SendMessage( m );
    }
    protected Boolean ReceiveMessage( Intercomunicador sender, String m ){
        if( !on ) return false;
        return super.ReceiveMessage(sender, m);
    }
    public Boolean OnOff(){
        on = !on;
        return on;
    }
    public Boolean on() {
        return on;
    }
}

```

```

    public String toString(){
        return super.toString() + " on:" + on;
    }
}

public class HomePhone extends TwoWay implements Fixed {
    private String address;
    public HomePhone( String serialNum, String num, Exchange e, String addr
){
        super( serialNum, num, e );
        SetAddress( addr );
    }
    public String toString() {
        return super.toString()+" A:"+address;
    }
    public void SetAddress( String addr ) {
        address = addr;
    }
}

```

### 3. Ejercicios.

#### 3.1 Ejercicio 1

- Implemente la Herencia y las interfaces para los casos de estudio Interfono y Celular.

#### 3.2 Ejercicio 2

- Realice una clase de prueba donde ponga en uso cada uno de los métodos de las clases desarrolladas.

#### 3.3 Ejercicio 3

- Tome provecho del polimorfismo y cree un método para que funcione con cualquier tipo de dispositivo TwoWay y uno mas para que funcione con cualquier objeto de tipo Mobile.

```

private static void JugarConWalkieTalkies() {
    WalkieTalkie [] walkieTalkies = {
        new WalkieTalkie( "WT1111" ),
        ::
    };
    for( int i=0; i<4; i++ )
        for( int j=0; j<4; j++ )
            if( i != j )
                walkieTalkies[i].AddListener( walkieTalkies[j] );
    PrenderMobiles( walkieTalkies );
};
JugarConOneWay( walkieTalkies[0], walkieTalkies[1],
                walkieTalkies[2], walkieTalkies[3] );}

```

Envía walkieTalkies

```

private static void JugarConOneWay( OneWay a, OneWay b, OneWay c,
                                    OneWay d ) {
    a.Push(); a.SendMessage("Hola ¿ya terminaste de bañarte?");a.Release();
    b.Push(); b.SendMessage("Ya...¿porqué?"); b.Release();
    a.Push(); a.SendMessage("Ya está tu cena..."); a.Release();
    ::
}

```

Recibe OneWay