Dependency Injection

Felipe Moreno, Augusto Pecho

July 4, 2016

Inyección de Dependencias (DI)

Introducción

Definiciones

Algunos casos

Tipos

Tipos - Ejemplos

Implementación

Introducción

Es un patrón de diseño y desarrollo de software que implementa la abstracción separando la creación de las dependencias de un cliente del comportamiento de un cliente. La dependencia trabaja de manera similar que un paso de parámetros. Una **dependencia** es un objeto que se puede utilizar (un servicio). Una **inyección** es el paso de una dependencia de un objeto dependiente (un cliente) que lo usaría.

Este patrón permite la abstracción de un programa cliente (objeto o clase) que requiera información de otro programa (objeto o clase) servidor, el cual enviará peticiones mediante inyecciones de código al programa cliente debido a que el construye el servicio y este cliente no sabrá ni necesita saber sobre el código de inyección, no necesita saber cómo se levanta o construye los servicios. Este patrón tuvo sus primeros usos en el módulo de **contenedor de inversión de control** que era los inicios de inyección de dependencias en su libro "Expert One-on-One J2EE Design and Development (Wrox Press, octubre 2002)".

Definiciones

- **servicio** es cualquier objeto que se puede usar (Dependencia).
- **cliente** es cualquier objeto que utiliza otros objetos (objeto dependendiente).
- **interfaces** es el que define como un cliente puede usar las dependencias (servicios).
- **inyector** es el que introduce los servicios al cliente, se encarga de la construcción de los servicios e inyectarlos al cliente.

Algunos casos

A nivel de usuario:

- Usted no necesita conocer el numero de contacto o número de las agencias, el departamento de administración te llama cuando es necesario.
- Usted no necesita conocer el método en como se pasa la voz en una llamada telefónica o como se maneja la entrada de datos por medio de un teclado, tal cual hace los sistemas linux, instalan el driver del controlador cualsea que se conecte y el usuario solo lo utiliza.

Entonces se puede decir de estos casos que los servicios se les proporcionan a los usuarios de una manera tal que si el servicio se vuelve a ajustar o se cambia, no es necesario modificar algo en los usuarios para que sigan manteniendo la funcionalidad.

Tipos

- **Constructor injection**: Las DI se proporcionan a través de un constructor de la clase.
- **Setter injection**: El cliente posee un método de seteo (setMethod) que el inyector utiliza para inyectar la dependencia.
- **Interface injection**: La DI proporciona un método de inyección que se inyectará en cualquier cliente que llama al método.

Tipos - Ejemplos

- **Constructor injection**:
  Client(Service service) {
  this.service = service;
  }

- **Setter injection**:
  // Setter method
  public void setService(Service service) {
  this.service = service;
  }

- **Interface injection**:

```
// clase interface
public interface ServiceSetter {
public void setService(Service service); }
//clase Cliente
public class Client implements ServiceSetter {
// Instancia desde el cliente.
private Service service;
// Método setter del cliente
@Override
public void setService(Service service) { this.service = service; } }
```

Implementación

- Se utilizó Android Studio 1.5.1
- Creación de la Interfaz

```
1  public interface OnLoginListener {
2      void onLogin(String usuario, String password
           );
3  }
```

- Creación del Controlador:

```
1  public class ControlLogin extends LinearLayout {
2          private OnLoginListener listener;
3          ......
4  }
```

- Creamos algunos métodos para relacionar con la Interfaz:

```
1  public void setOnLoginListener(OnLoginListener l
       ) {
2      listener = l;
3  }
4
5  private void asignarEventos() {
6      btnLogin.setOnClickListener(new
           OnClickListener() {
7          @Override
8          public void onClick(View v) {
9              listener.onLogin(txtUsuario.getText
                   ().toString(),
```

```
10                              txtPassword.getText().
                                    toString());
11              }
12          });
13      }
14
15      public void setMensaje(String msg) {
16          lblMensaje.setText(msg);
17      }
```

- Creamos la clase Principal:

```
1  public class Principal extends AppCompatActivity
       {
2      private ControlLogin ctlLogin;
3
4      @Override
5      protected void onCreate(Bundle
           savedInstanceState) {
6          super.onCreate(savedInstanceState);
7          setContentView(R.layout.
               activity_principal);
8          Toolbar toolbar = (Toolbar) findViewById
               (R.id.toolbar);
```

```
9           setSupportActionBar(toolbar);
10          ctlLogin = (ControlLogin)findViewById(R.
                id.CtlLogin);
11
12          ctlLogin.setOnLoginListener(new
                OnLoginListener() {
13              @Override
14              public void onLogin(String usuario,
                    String password) {
15                  //Validamos el usuario y
                        password
16                  if (usuario.equals("demo") &&
                        password.equals("demo"))
17                      ctlLogin.setMensaje("Login
```

```
                                correcto!");
18                        else
19                            ctlLogin.setMensaje("Vuelva
                                a intentarlo.");
20                    }
21                });
22            }
23            .........
24            .........
25    }
```

Gracias