# Annotation et Pattern

## Pattern d'injection

Nous allons regarder le pattern d'injection.

Soit une interface Store permettant de sauvegarder des données dans un fichier

```
public interface Store {

	public void initStore(String str);
	public void storeData(String filename, byte [] data);
	
}
```

## Premiere implémentation.

La première implémentation utilise la librairie Apache common pour sauvegarder des données dans un fichier

```
import org.apache.commons.io.IOUtils;

public class PlainStore implements Store {

	public void initStore(String str) {
		// TODO Auto-generated method stub
		
	}

	public void storeData(String filename, byte[] data) {
		try{
```

```
        FileOutputStream stream=new FileOutputStream(filename);

        IOUtils.write(data, stream);
        stream.close();
        }
        catch (Exception e)
        {
        throw new RuntimeException(e);
        }
    }

}
```

# Deuxième Implementation

La deuxième implémentation utilise de la cryptographie pour enregistrer les données

```
package be.etnic.guice;

import java.io.FileOutputStream;
import java.security.SecureRandom;

import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.SecretKeySpec;

import org.apache.commons.io.IOUtils;

public class AESStoreProcessor implements Store{

    private SecretKey key;

    public void initStore(String str) {
        try{
            SecureRandom sr = SecureRandom.getInstanceStrong();
            byte[] salt = new byte[16];
```

```
    sr.nextBytes(salt);

    PBEKeySpec spec = new PBEKeySpec(str.toCharArray(), salt, 65536, 128);
    SecretKey tmp = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1")
        .generateSecret(spec);
    this.key = new SecretKeySpec(tmp.getEncoded(), "AES");

  }
  catch (Exception e)
  {
    throw new RuntimeException(e);
  }
}

public void storeData(String filename, byte[] data) {
  try{
    Cipher aes = Cipher.getInstance("AES");
    aes.init(Cipher.ENCRYPT_MODE, key);
   FileOutputStream stream=new FileOutputStream(filename);

  IOUtils.write(aes.doFinal(data)
    , stream);
  stream.close();
  }
  catch (Exception e)
  {
    throw new RuntimeException(e);
  }
}

}
```

# Module de haut niveau.

Nous mettons a disposition une classe permettant d'avoir des store

```
public class StoreProcessor {

	Store store;

	public StoreProcessor(Store store)
	{
		this.store=store;
	}
	public void storeData(String filename,String initParameter,byte [] data)
	{
		store.initStore(initParameter);
		store.storeData(filename, data);
	}
}
```

Le hic est qu'il faut passer un store en parametre du store processor.

On peut utiliser ici un pattern d'injection

```
public class StoreModule extends AbstractModule {
  @Override
  protected void configure() {

    /*
     * This tells Guice that whenever it sees a dependency on a TransactionLog,
     * it should satisfy the dependency using a DatabaseTransactionLog.
     */
   bind(Store.class).to(PlainStore.class);
  }

}
```

Et placer l'injection sur le constructeur de StoreProcessor

```
public class StoreProcessor {
	Store store;

	@Inject
	public StoreProcessor(Store store)
	{
```

```
    this.store=store;
  }
  public void storeData(String filename,String initParameter,byte [] data)
  {
    store.initStore(initParameter);
    store.storeData(filename, data);
  }
}
```

Dans la fonction main est ainsi faites

```
public class App
{
    public static void main( String[] args )
    {
        System.out.println( "Hello World!" );
        StoreProcessor storeProcessor=new StoreProcessor(new AESStoreProcessor());
        storeProcessor.storeData("test.txt", "pilou", "hello world".getBytes());
        Injector injector = Guice.createInjector(new StoreModule());
        GuiceStoreProcessor storeProcessor2=injector.getInstance(GuiceStoreProcessor.class);
        storeProcessor2.storeData("test2.txt", "pilou", "hello world".getBytes());

    }
}
```

---

Revision #1
Created 11 November 2019 13:34:51 by Admin
Updated 11 November 2019 14:03:13 by Admin