Tapestry5HowToDecorateService

This is an example of a simple service decorator.

Documentation and a more complicated example are here

In this example we will create two simple services: Service and AnotherService.

We will create a decorator(interceptor) that will call a method in Service before and after calling "run" method in AnotherService.

First we declare the services in service bind method so they are created automaticaly.

then we create an interceptor, since we implement our interceptor manualy we don't need a complex decorator like the loging decorator in documentation.

We can cast the delegate object into `AnotherService` because we will use this only to intecept that specific service.

WARNING: If, instead of casting the `delegate` object, we declare `delegate` as `AnotherService` tapestry will ignore the `decorateAnotherService` decorator method.

Appmodule looks like this:

```
public class AppModule{
    public static void bind (ServiceBinder binder){
        binder.bind(Service.class, ServiceImpl.class);
        binder.bind(AnotherService.class, AnotherServiceImpl.class);
    }

   public static AnotherService decorateAnotherService(
           Object delegate, Service service, Log log)
   {
       return new Interceptor((AnotherService)delegate, service, log );
   }
}
```

The interceptor is actualy an implementation of the `AnotherService` interface(this way usage of proxies is avoided).

The interceptor created here uses reference to `AnotherService` which can be the original service or another delegate

In our example Interceptor will call `Service.execute()` before and after `AnotherService.run()`.

```
public class Interceptor implements AnotherService{

    private final Service service;
    private final Log log;
    private final AnotherService delegate;

    public Interceptor(AnotherService delegate,Service service, Log log) {
        this.delegate = delegate;
        this.service = service;
        this.log = log;
    }

    public void run() {
        // Logic before delegate invocation here.
        log.info("<<< Before delegate >>>");
        service.execute();

        delegate.run();

        service.execute();
        // Logic after delegate invocation here.
        log.info("<<< After delegate >>>");
    }
 }
```

Service and ServiceImpl

```
public interface Service {
    public void execute();
}

public class ServiceImpl implements Service {
    private Log log;

    public ServiceImpl(Log log) {
        this.log = log;
    }

    public void execute() {
        log.info("<-- Inside Service method -->");
    }
}
```

AnotherService and AnotherServiceImpl

```
public interface AnotherService {
    public void run();
}

public class AnotherServiceImpl implements AnotherService{
    Log log;

    public AnotherServiceImpl(Log log) {
        this.log = log;
    }

    public void run(){
        log.info("Inside another service");
    }
}
```

After all this, you can Inject the `AnotherService` into your page

```
    @Inject private AnotherService another;
```

after you call `another.run()` somewhere in your page, the interceptor will be called instead of original service and you will see similar lines in your log file:

```
11:28:28.390 INFO Interceptor.run(Interceptor.java:19) >52> <<< Before delegate >>>
11:28:28.390 INFO ServiceImpl.execute(ServiceImpl.java:14) >54> <-- Inside Service method -->
11:28:28.390 INFO AnotherServiceImpl.run(AnotherServiceImpl.java:13) >53> Inside another service
11:28:28.390 INFO ServiceImpl.execute(ServiceImpl.java:14) >54> <-- Inside Service method -->
11:28:28.390 INFO Interceptor.run(Interceptor.java:26) >52> <<< After delegate >>>
```