08

Mãos a obra: Observer

Vamos utilizar o padrão `Observers` para poder desacoplar nosso `PhaseListener` das classes que estão interessadas em saber se estamos antes ou depois de uma determinada fase.

O **CDI** nos ajuda a implementar esse padrão, basta criarmos uma classe que recebe o objeto que queremos observar e anotar esse objeto com `@Observer`. Para o **CDI** o objeto que estamos observando é chamado de evento.

```
public class LogPhase {

    public void log( @Observes PhaseEvent phaseEvent){
        System.out.println("FASE: " + phaseEvent.getPhaseId());
    }

}
```

Além disso podemos qualificar o evento que estamos observando.

```
public class LogPhase {

    public void log( @Observes @Before  @Phase(Phases.RESTORE_VIEW) PhaseEvent phaseEvent){
        System.out.println("FASE: " + phaseEvent.getPhaseId());
    }

}
```

Vamos criar então os qualificadores para que o projeto `Livraria` compile. E depois disso vamos criar alguém que dispara esse evento.

No projeto livraria crie as anotações `@Before`, `@After` e `@Phase`:

```
@Qualifier
@Target(ElementType.PARAMETER)
@Retention(RetentionPolicy.RUNTIME)
public @interface Before {}
```

```
@Qualifier
@Target(ElementType.PARAMETER)
@Retention(RetentionPolicy.RUNTIME)
public @interface After {}
```

```
@Qualifier
@Target(ElementType.PARAMETER)
@Retention(RetentionPolicy.RUNTIME)
```

```
public @interface Phase {

    Phases value();


    enum Phases{
        RESTORE_VIEW(PhaseId.RESTORE_VIEW),
        APPLY_REQUEST_VALUES(PhaseId.APPLY_REQUEST_VALUES),
        PROCESS_VALIDATIONS(PhaseId.PROCESS_VALIDATIONS),
        UPDATE_MODEL_VALUES(PhaseId.UPDATE_MODEL_VALUES),
        INVOKE_APPLICATION(PhaseId.INVOKE_APPLICATION),
        RENDER_RESPONSE(PhaseId.RENDER_RESPONSE);

        private PhaseId phaseId;

        Phases(PhaseId phaseId){
            this.phaseId = phaseId;
        }

        public PhaseId getPhaseId() {
            return phaseId;
        }
    }

}
```

Agora vamos alterar a classe `PhaseListenerGenerico` para que ele notifique em que momento e qual fase está no momento.

```
public class PhaseListenerGenerico implements PhaseListener{


    private static final long serialVersionUID = 4332180564534271099L;

    private PhaseListernerObserver observer = new PhaseListernerObserver();

    @Override
    public void afterPhase(PhaseEvent event) {
        observer
            .after()
            .fire(event);

    }

    @Override
    public void beforePhase(PhaseEvent event) {
        observer
            .before()
            .fire(event);
    }

    @Override
    public PhaseId getPhaseId() {
        return PhaseId.ANY_PHASE;
    }
```

```
}
```

Como estamos criando um `PhaseListener` em um projeto que não usa o `JSF` não podemos injetar dependências nele (se esse mesmo `PhaseListener` fosse criado no projeto `Livraria` funcionaria a injeção de dependência, pois o projeto `Livraria` é um projeto configurado para usar o `JSF`).

Vamos agora criar a classe `PhaseListenerObserver`

```
public class PhaseListernerObserver {

    private Annotation momento;


    public PhaseListernerObserver after(){
        this.momento = new AnnotationLiteral<After>(){};
        return this;
    }


    public PhaseListernerObserver before(){
        this.momento = new AnnotationLiteral<Before>(){};
        return this;
    }

    public void fire(PhaseEvent phaseEvent){

    }

}
```

Usamos aqui a classe `AnnotationLiteral` para podermos armazenar qual o qualificador para o momento (`Before` ou `After`) devemos usar para disparar o evento. Porém essa mesma estratégia não pode ser usada para a anotação `Phase` pois ela espera receber um valor para informar qual a fase estamos querendo observar.

Então iremos criar uma classe que estende de `AnnotationLiteral` e especificar qual a fase essa anotação se refere.

```
public class PhaseLiteral extends AnnotationLiteral<Phase> implements Phase {



    private static final long serialVersionUID = 3320747441506123437L;

    private Phases phases;

    public PhaseLiteral(PhaseId phaseId) {
        phases = Phase.Phases.valueOf(phaseId.getName());
    }

    @Override
    public Phases value() {
```

```
            return phases;
        }

    }
```

Dessa forma informamos que a classe `PhaseLiteral` é uma `AnnotationLiteral` da anotação `Phase`, além disso implementamos a própria anotação `Phase` para podermos sobrescrever o método que retorna qual fase estamos interessado em observar.

Criamos um construtor que recebe um objeto `PhaseId` e retornamos qual a fase estamos interessado a partir desse objeto.

Agora basta modificar a classe `PhaseListenerObserver` para usar o qualificador `PhaseLiteral` e disparar o evento.

```
public class PhaseListernerObserver {

    private BeanManager observer = CDI.current().getBeanManager();
    private Annotation momento;


    public PhaseListernerObserver after(){
        this.momento = new AnnotationLiteral<After>(){};
        return this;
    }


    public PhaseListernerObserver before(){
        this.momento = new AnnotationLiteral<Before>(){};
        return this;
    }

    public void fire(PhaseEvent phaseEvent){
        observer.fireEvent(phaseEvent, momento,new PhaseLiteral(phaseEvent.getPhaseId()) );
    }

}
```

Estamos usando o `BeanManager` pois não podemos injetar nossas dependências no `PhaseListenerGenerico`, se pudéssemos usar injeção de dependências nossa classe ficaria assim:

```
public class PhaseListernerObserver {

    @Inject
    private Event<PhaseEvent> observer;
    private Annotation momento;


    public PhaseListernerObserver after(){
        this.momento = new AnnotationLiteral<After>(){};
        return this;
    }


    public PhaseListernerObserver before(){
        this.momento = new AnnotationLiteral<Before>(){};
```

```
            return this;
        }

        public void fire(PhaseEvent phaseEvent){
            observer.
                    select(momento).
                    select(new PhaseLiteral(phaseEvent.getPhaseId())).
                    fire(phaseEvent);
        }

    }
```

Feito isso vamos instalar nosso `jar` no repositório local: Clique com o botão direito do mouse sobre o projeto `Alura-lib` e selecione `Run as > Maven Install` .

Depois vamos no projeto `Livraria` e vamos atualiza-lo: Clique com o botão direito do mouse sobre o projeto `Livraria` e selecione `Maven > Update project...` .

Corrija os `import` s da classe `LogPhase` e verifique se tudo continua funcionando.