Agile Development with CDI in a Java EE World

R. Turini

Java EE Instructor

VRaptor lead developer

Tech book author

caelum

ensino e inovação

Chico Sokol

caelum
ensino e inovação

caelum

ensino e inovação

Java training

Community

Open Source!

What is

VRaptor ?

MVC

web framework

Action-based

Why VRaptor?

Simple

Flexible

Opensource

CDI

CDI

+ JavaEE 7

VRaptor

basics

```
@Controller
public class SpeakerController {

    @Inject private SpeakerDao speakerDao;
    @Inject private Result result;

    @Get("speaker/add")
    public void add() {}

    @Post("speaker/add")
    public void add(Speaker speaker) {
        speakerDao.save(speaker);
        result.redirectTo(this).list();
    }

    public void list() {
        result.include("speakers", speakerDao.list());
    }
}
```

Serious

CoC

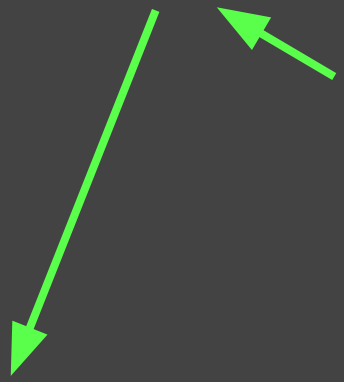
```
@Controller
public class SpeakerController {

    public void add() {
    }
}
```

```
@Controller
public class SpeakerController {

    public void add() {
    }
}
```

speaker/add

pattern: controllerName/methodName

```
@Controller
public class SpeakerController {

    public void add() {
    }
}
```



speaker/add

pattern: controllerName/methodName

/WEB-INF/jsp/speaker/add.jsp

pattern: /WEB-INF/jsp/controllerName/methodName.jsp

Methods with the

same name ?

```
@Controller
public class SpeakerController {


    public void add() {
    }


    public void add(Speaker speaker) {
        speakerDao.save(speaker);
    }
}
```

URL

ambiguity

```
@Controller
public class SpeakerController {


    public void add() {
    }


    public void add(Speaker speaker) {
        speakerDao.save(speaker);
    }
}
```

```
@Controller
public class SpeakerController {

    @Path("/speaker/form")
    public void add() {
    }

    @Path("/speaker/add")
    public void add(Speaker speaker) {
        speakerDao.save(speaker);
    }
}
```

explicitly defining the path, using:

```
@Path("/your/custom/path")
```

```
@Controller
public class SpeakerController {

    @Get
    public void add() {
    }

    @Post
    public void add(Speaker speaker) {
        speakerDao.save(speaker);
    }
}
```

limit access to a controller method acording with the HTTP verb, using:

`@Get`, `@Post`, `@Put` or `@Delete`

```
@Controller
public class SpeakerController {

    @Get("/speaker/form")
    public void add() {
    }

    @Post("/speaker/add")
    public void add(Speaker speaker) {
        speakerDao.save(speaker);
    }
}
```

Or both!

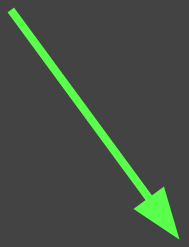
```
@Get("/your/custom/path")
```

Parameter

injection

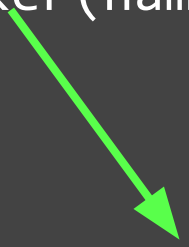
```
@Controller
public class SpeakerController {

    // other methods

    @Post("/speaker/add")
    public void add(Speaker speaker) {
        speakerDao.save(speaker);
    }
}
```

```
@Controller
public class SpeakerController {

    // other methods

    @Post("/speaker/add")
    public void add(Speaker speaker) {
        speakerDao.save(speaker);
    }
}
```



Where is this coming from?

```
@Controller
public class SpeakerController {

    // other methods

    @Post("/speaker/add")
    public void add(String name, int age) {
        Speaker speaker = new Speaker(name, age);
        speakerDao.save(speaker);
    }
}
```



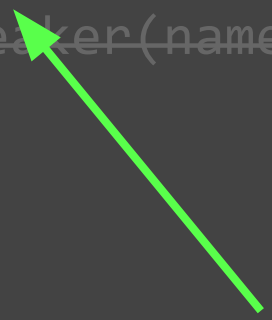
Where is this coming from?

```
@Controller
public class SpeakerController {

    // other methods

    @Post("/speaker/add")
    public void add(String name, int age) {
        Speaker speaker = new Speaker(name, age);
        speakerDao.save(speaker);
    }
}
```

```
<form action="/speaker/add" method="post">
    Name: <input name="name">
    Age: <input name="age">
    <input type="submit">
</form>
```

```
@Controller
public class SpeakerController {

    // other methods

    @Post("/speaker/add")
    public void add(Speaker speaker) {
        Speaker speaker = new Speaker(name, age);
        speakerDao.save(speaker);
    }
}
```

```java
@Controller
public class SpeakerController {

    // other methods

    @Post("/speaker/add")
    public void add(Speaker speaker) {
        Speaker speaker = new Speaker(name, age);
        speakerDao.save(speaker);
    }
}
```



```html
<form action="/speaker/add" method="post">
    Name: <input name="speaker.name">
    Age: <input name="speaker.age">
    <input type="submit">
</form>
```

```
Name: <input name="speaker.name">
Age: <input name="speaker.age">
```

```
?speaker.name=Chico&speaker.age=26
```

```
new Speaker("Chico", 26)
```

```
Name: <input name="speaker.name">
```

?speaker.name=Chico

```
speaker.setName("Chico")
```

Redirecting

to another logic

```
@Controller
public class SpeakerController {



    @Post("/speaker/add")
    public void add(Speaker speaker) {
        speakerDao.save(speaker);
        // must redirect to list method
    }


    public void list() {
       // include something to the view
    }
}
```

```
@Controller
public class SpeakerController {



    @Post("/speaker/add")
    public void add(Speaker speaker) {
        speakerDao.save(speaker);
        result.redirectTo("/speaker/list");
    }


    public void list() {
       // including something to the view
    }
}
```

```
@Controller
public class SpeakerController {



    @Post("/speaker/add")
    public void add(Speaker speaker) {
        speakerDao.save(speaker);
        result.redirectTo(this).list();
    }


    public void list() {
       // including something to the view
    }
}
```

DI

```
@Controller
public class SpeakerController {

    @Inject private Result result;

    @Post("/speaker/add")
    public void add(Speaker speaker) {
        speakerDao.save(speaker);
        result.redirectTo(this).list();
    }


    public void list() {
       // including something to the view
    }
}
```

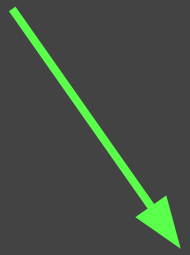
```
@Controller
public class SpeakerController {

    @Inject private Result result;

    @Post("/speaker/add")
    public void add(Speaker speaker) {
        speakerDao.save(speaker);
        result.redirectTo(this).list();
    }


    public void list() {
       // including something to the view
    }
}
```

including

attributes

```
@Controller
public class SpeakerController {

    @Inject private Result result;

    @Post("/speaker/add")
    public void add(Speaker speaker) {
        speakerDao.save(speaker);
        result.redirectTo(this).list();
    }

    @Get("speaker/list")
    public void list() {
       result.include("speakers", speakerDao.list());
    }
}
```

```
public void list() {
   result.include("speakers", speakerDao.list());
}
```



```
<html>
  <body>
    <ul>
      <c:forEach items="${speakers}" var="speaker">
        <li>${speaker.name}</li>
      </c:forEach>
    </ul>
  </body>
</html>
```

what about

SERVICES ?

```
public void list() {
    result.use(Results.json())
        .from(speakerDao.list()).serialize();
}
```

```
{"list":[
    {"name":"Rodrigo Turini", ...},
    {"name":"Chico Sokol", ...}
]}
```

```
public void list() {
    result.use(Results.xml())
        .from(speakerDao.list()).serialize();
}
```

```
<list>
    <speaker>
        <name>Rodrigo Turini</name> ...
    </speaker>
    <speaker>
        <name>Chico Sokol</name> ...
    </speaker>
</list>
```

Taking advantage of

CDI

Inject

ANY

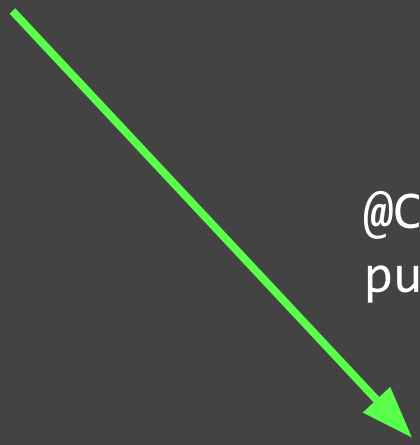
bean

```
public class SpeakerDao {

    public void save(Speaker speaker) {
        //persist our speaker
    }

    public List<Speaker> list() {
        //return all speakers from db
    }
}
```

```
@RequestScoped
public class SpeakerDao {

    public void save(Speaker speaker) {
        //persist our speaker
    }

    public List<Speaker> list() {
        //return all speakers from db
    }
}
```

Any annotated class is a CDI Bean

```java
@RequestScoped
public class SpeakerDao {
    //...
}
```



```java
@Controller
public class SpeakerController {

    @Inject
    private SpeakerDao speakerDao;

    @Post("/speaker/add")
    public void add(Speaker speaker) {
        speakerDao.save(speaker);
    }
}
```

Inject

third party

classes

```
@RequestScoped
public class SpeakerDao {

    private EntityManager em;

    public void save(Speaker speaker) {
        em.persist(speaker);
    }
}
```

```
@RequestScoped
public class SpeakerDao {
    @Inject
    private EntityManager em;

    public void save(Speaker speaker) {
        em.persist(speaker);
    }
}
```

```
@RequestScoped
public class SpeakerDao {
    @Inject
    private EntityManager em;

    public void save(Speaker speaker) {
        em.persist(speaker);
    }
}
```

EntityManager isn't a CDI bean!

```
@Produces @RequestScoped
public EntityManager produce() {
    return emf.createEntityManager();
}
```

```
@ApplicationScoped
public class EntityManagerProducer {

    private EntityManagerFactory emf;

    @PostConstruct
    public void init() {
      emf = // initializing EM factory
    }

    @Produces @RequestScoped
    public EntityManager produce() {
        return emf.createEntityManager();
    }
}
```

what about

TRANSACTIONS?

```
@RequestScoped
public class SpeakerDao {

    @Inject private EntityManager em;

    public void save(Speaker speaker) {
        em.getTransaction().begin();
        em.persist(speaker);
        em.getTransaction().commit();
    }
}
```

CDI's Interceptor

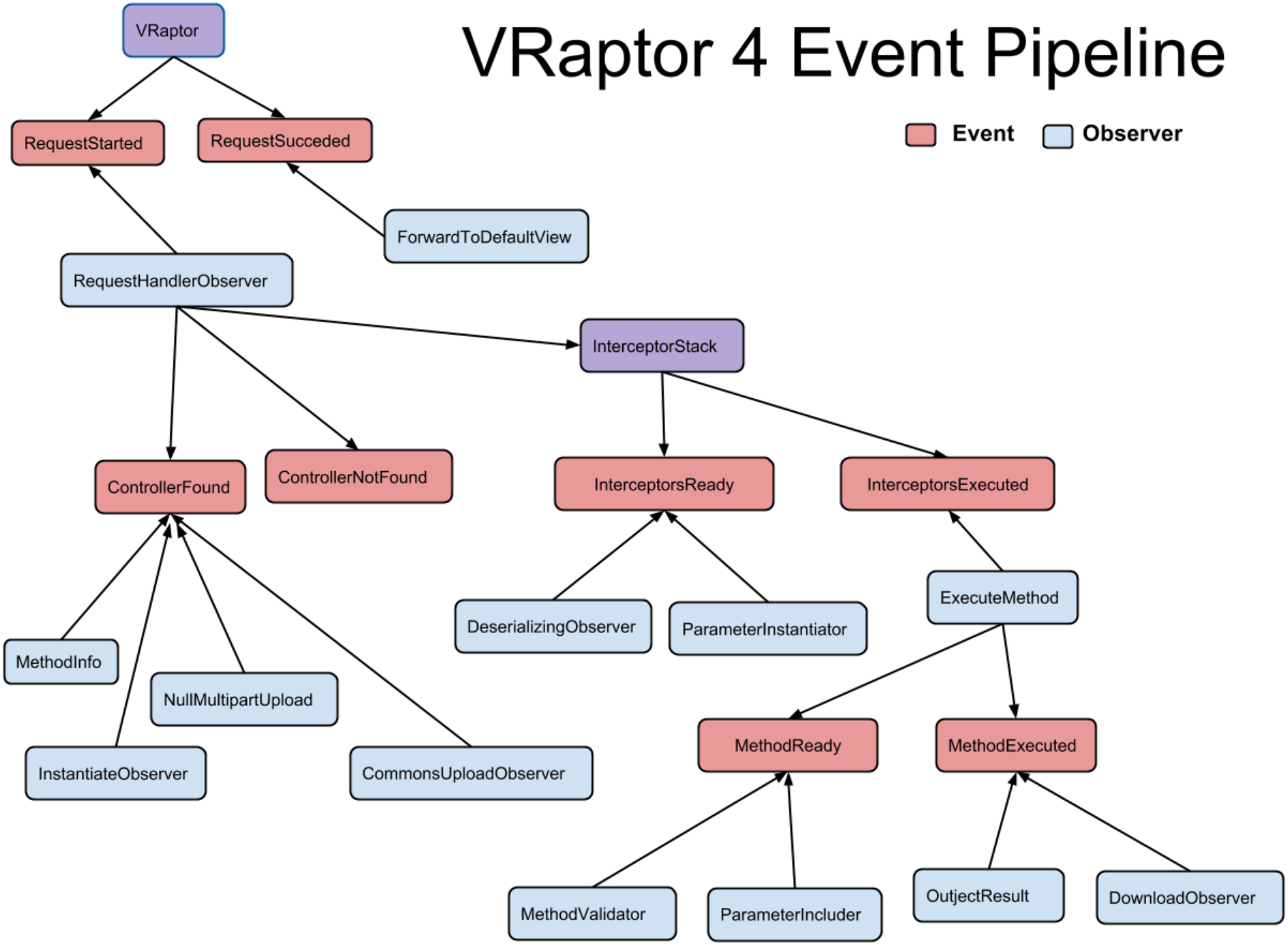
```
@InterceptorBinding
@Target({METHOD,TYPE})
@Retention(RUNTIME)
public @interface UsingTransaction {
}
```

```
@Interceptor @UsingTransaction
public class TransactionalInterceptor {

    @Inject private EntityManager em;

    @AroundInvoke
    public Object intercept(InvocationContext context)
      throws Exception {

        em.getTransaction().begin();
        Object proceed = context.proceed();
        em.getTransaction().commit();
        return proceed;
    }
}
```

```
@RequestScoped
public class SpeakerDao {

    @Inject private EntityManager em;


    public void save(Speaker speaker) {
        em.getTransaction().begin();
        em.persist(speaker);
        em.getTransaction().commit();
    }
}
```

```
@RequestScoped
public class SpeakerDao {

    @Inject private EntityManager em;

    @UsingTransaction
    public void save(Speaker speaker) {
        em.getTransaction().begin();
        em.persist(speaker);
        em.getTransaction().commit();
    }
}
```

CDI's Events


VRaptor 4 Event Pipeline
Event
Observer
VRaptor
RequestStarted
RequestSucceded
ForwardToDefaultView
RequestHandlerObserver
InterceptorStack
ControllerFound
ControllerNotFound
InterceptorsReady
InterceptorsExecuted
ExecuteMethod
DeserializingObserver
ParameterInstantiator
MethodInfo
NullMultipartUpload
InstantiateObserver
CommonsUploadObserver
MethodReady
MethodExecuted
MethodValidator
ParameterIncluder
OutjectResult
DownloadObserver

You can observe

VRaptor

events

```
@ApplicationScoped
public class VRaptorStartupObserver {

    public void init(@Observes VRaptorInitialized event){

        // send email notifying deploy

    }
}
```

```
@ApplicationScoped
public class VRaptorStartupObserver {

    public void init(@Observes VRaptorInitialized event){

        // send email notifying deploy

    }
}
```

Serious

CoC

Serious

CoC

Specialize!

```
public class MyPathResolver {



    protected String getPrefix() {
        return "/WEB-INF/pages/";
    }
}
```

```
public class MyPathResolver extends DefaultPathResolver {

    // delegate constructor

    @Override
    protected String getPrefix() {
        return "/WEB-INF/pages/";
    }
}
```

```
@Specializes
public class MyPathResolver extends DefaultPathResolver {

    // delegate constructor

    @Override
    protected String getPrefix() {
        return "/WEB-INF/pages/";
    }
}
```

PROGRAMMATIC

configuration

specialize

anything!

(thanks to global activation)

@Decorator @Named @Priority

@Qualifier @Alternative

any CDI feature

YOU WANT!

VRaptor

CDI

JavaEE

Bean Validation

```
public class Speaker {

    private String name;

    private String email;

    private int age;

    // getters and setters
}
```

```
public class Speaker {

    @NotNull @NotEmpty
    private String name;

    @NotNull @Pattern(regexp="...")
    private String email;

    @Min(0) @Max(100)
    private int age;


    // getters and setters
}
```

Bean Validation

+

VRaptor

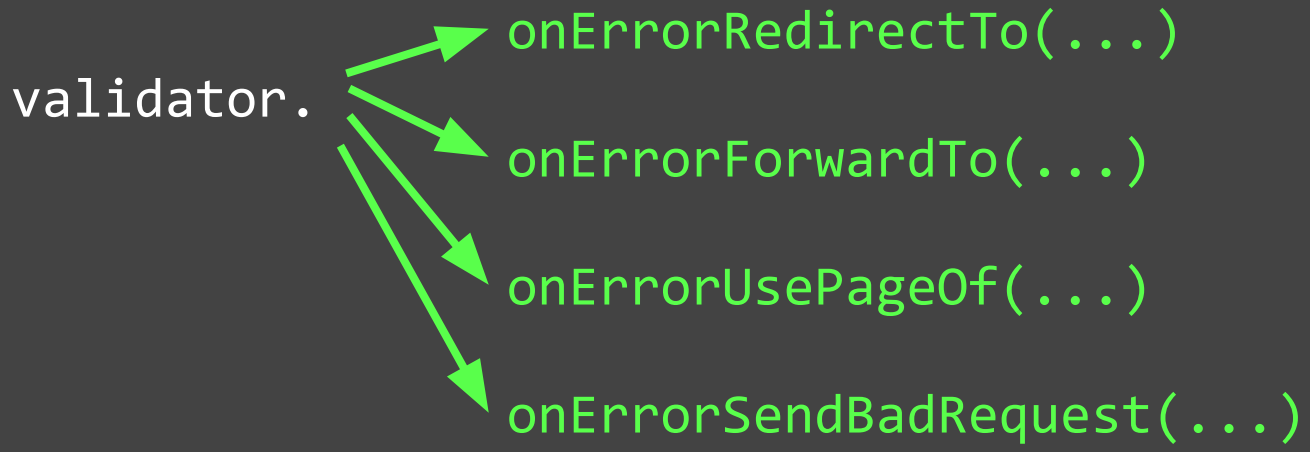
```
@Controller
public class SpeakerController {

    //...

    public void add(Speaker speaker) {

        speakerDao.save(speaker);
        result.redirectTo(this).list();
    }
}
```

```
@Controller
public class SpeakerController {

    //...

    public void add(@Valid Speaker speaker) {

        speakerDao.save(speaker);
        result.redirectTo(this).list();
    }
}
```

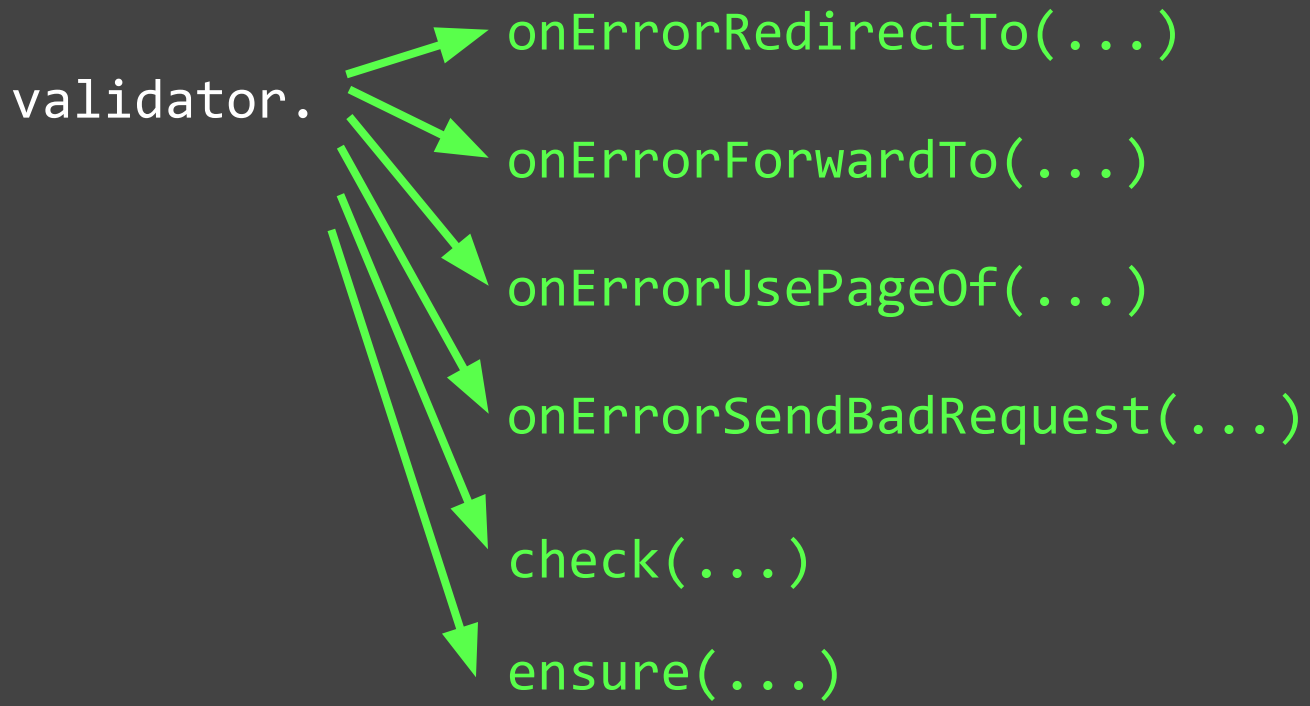
```
@Controller
public class SpeakerController {



    public void add(@Valid Speaker speaker) {
        //redirect in case of validation errors
        speakerDao.save(speaker);
        result.redirectTo(this).list();
    }
}
```

```
import br.com.caelum.vraptor.validator.Validator;

@Controller
public class SpeakerController {

    @Inject private Validator validator;

    public void add(@Valid Speaker speaker) {
        validator.onErrorRedirectTo(this).add();
        speakerDao.save(speaker);
        result.redirectTo(this).list();
    }
}
```

```
import br.com.caelum.validator.Validator;
```

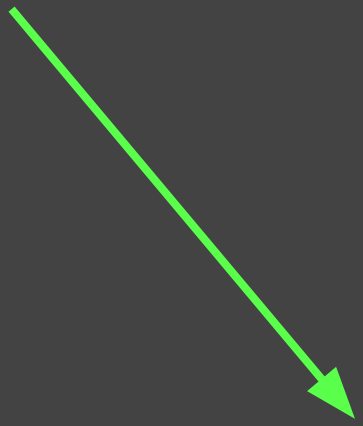


```
import br.com.caelum.validator.Validator;
```



JPA

```
@ApplicationScoped
public class EntityManagerProducer {

    @Inject private EntityManagerFactory emf;

     @RequestScoped @Produces
     public EntityManager produce() {
         return emf.createEntityManager();
     }

}
```



```
@RequestScoped
public class SpeakerDao {

    @Inject
    private EntityManager em;

}
```

```
@ApplicationScoped
public class EntityManagerProducer {

    @Inject private EntityManagerFactory emf;

     @RequestScoped @Produces
     public EntityManager produce() {
         return emf.createEntityManager();
     }
}
```

```
@RequestScoped
public class SpeakerDao {

    @PersistenceContext
    private EntityManager em;

}
```

```
@RequestScoped
public class SpeakerDao {

    @PersistenceContext
    private EntityManager em;

    public void save(Speaker speaker) {
        em.getTransaction().begin();
        em.persist(speaker);
        em.getTransaction().commit();
    }

    public List<Speaker> list() {
        return em
            .createQuery("select s from Speaker s")
            .getResultList();
    }
}
```

EJB

```
@RequestScoped @Stateless
public class SpeakerDao {

    @PersistenceContext
    private EntityManager em;

    public void save(Speaker speaker) {
        em.getTransaction().begin();
        em.persist(speaker);
        em.getTransaction().commit();
    }

    public List<Speaker> list() {
        return em
            .createQuery("select s from Speaker s")
            .getResultList();
    }
}
```

JTA

```
@RequestScoped @Stateless
public class SpeakerDao {

    @PersistenceContext
    private EntityManager em;

    @Transactional
    public void save(Speaker speaker) {
        em.getTransaction().begin();
        em.persist(speaker);
        em.getTransaction().commit();
    }

    public List<Speaker> list() {
        return em
            .createQuery("select s from Speaker s")
            .getResultList();
    }
}
```

JMS

Schedulers

JavaMail

and MORE!

any **JavaEE** *feature*

YOU WANT!

github.com/caelum/vraptor-javaone

what about

SERVLET CONTAINERS

Plugins!

vraptor-jpa

EntityManager injection and transaction support

vraptor-hibernate

Session injection and transaction support

vraptor-simplemail	E-mail delivering and templating
vraptor-quartzjob	Task scheduling

and also for a

MANAGED ENVIRONMENT

vraptor-actioncache

Caching of controller actions

vraptor-paginator

JPA/Hibernate query pagination

vraptor-i18n	Improved i18n
vraptor-biscotti	Type-safe i18n messages

vraptor-pannetone

Compiled and type-safe views

vraptor-brutauth

Complex authorization rules

more at

github.com/caelum/vraptor-contrib

Thank you

vraptor

 github.com/caelum/vraptor4

 questions.vraptor.org

 caelum-vraptor-en@googlegroups.com

 vraptor.org