Novelties in Java EE 6: Contexts and Dependency Injection for the Java™ EE Platform (CDI)

Trayan Iliev

IPT – Intellectual Products & Technologies
e-mail: tiliev@iproduct.org
web: http://www.iproduct.org

Trayan Iliev
IPT – Intellectual Products & Technologies

Survey

Which enterprise applications development platform(s) are you using today?

- Spring?
- J2EE™ (Java EE 1.4)?
- Java™ EE 5?
- Java™ EE 6?
- Other?
- Not yet ...

Trayan Iliev
IPT – Intellectual Products & Technologies

Java™ Platform, Enterprise Edition 6

- Java™ Platform Enterprise Edition is a specification developed by Oracle® (Sun) together with partners - BEA Systems, Borland, E.piphany, Hewlett-Packard, IBM, Inria, Novell, Red Hat, SAP, Sybase, Apache etc. to facilitate development of reliable, configurable, scalable, inter-operable, platform independent server applications and components using Java™ programming language
- Based on Java™ SE 6

Trayan Iliev
IPT – Intellectual Products & Technologies

Java™ Enterprise Edition 6 - Timeline:

Event	Original date	actual date	estimated date (*)
Expert group formed	Jul 2007	07 Oct, 2007	
First expert draft	Aug 2007	unknown	
Early Draft Review	Oct 2007	22 Nov, 2008	
Public Review	Dec 2007	23 Feb, 2009	
Proposed Final Draft	Mar 2008	06 Oct, 2009 – ???	
Final Approval Ballot	not originally planned	17 Nov, 2009-30 Nov 2009	
RI beta release	Q2 2008		Q4 2009
Final release	Q4 2008	10 Dec, 2009	
Implementation in Glassfish	10 Dec, 2009		
Implementation in Jboss	17 Dec, 2010		H1 2010 (Final release + several months)

Source: http://jdevelopment.nl/open-source/java-ee-6-progress-page/

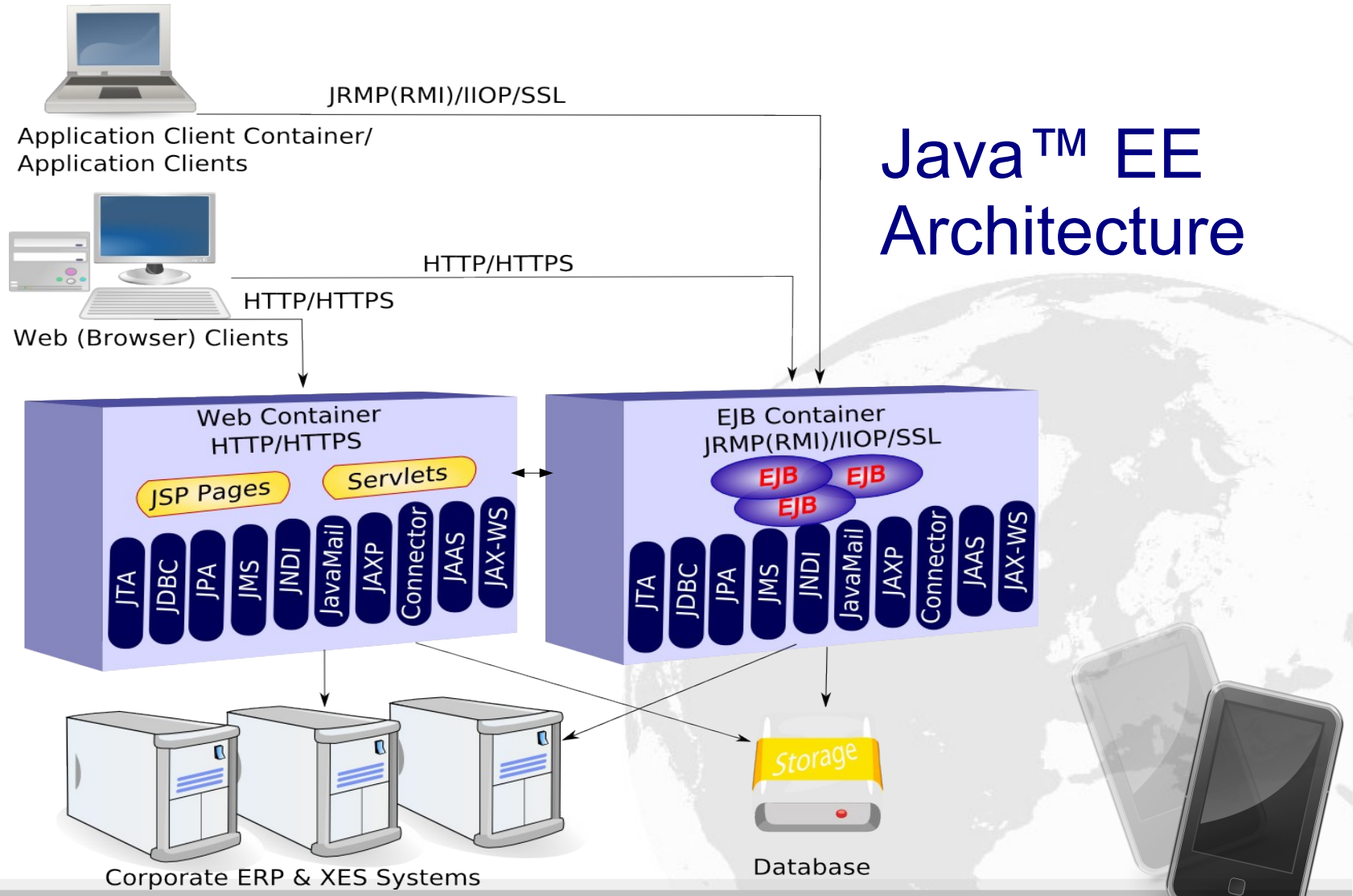
Trayan Iliev
IPT – Intellectual Products & Technologies

Java™ Platform, Enterprise Edition

- Java™ EE defines:
  - Programming models for application development and Application Programming Interfaces (APIs) for building distributed, multi-component, applications, as well as their packaging and installation
  - A stack of integrated services and APIs available out-of-the-box to speed-up the development, lower complexity, and improve productivity
  - Common logical architecture integrating different types of components and component containers

Trayan Iliev
IPT – Intellectual Products & Technologies

Java™ EE Architecture



Trayan Iliev
IPT – Intellectual Products & Technologies

Main Tiers in Java™ EE Architecture

- **Client tier** – including components that execute on client's computer (command line or Graphical User Interface clients)
- **Web tier** – including web components (Servlets, JSP, Facelets, etc.), and web services (SOAP, REST), which are executed in the web container of the Java EE server
- **Business tier** – business components: Enterprise Java Beans (EJB), Plain Old Java Objects (POJO – Java Beans), Java Persistence API (JPA) Entities
- **Enterprise Information System (EIS) tier** - including external information systems (ERP, XES) and databases accessible through standardized connectors (Java Connector Architecture - JCA)

Services Provided by Java™ EE Container (1)

- JavaEE containers provide necessary lower level infrastructure services to allow developers to focus on business and presentation logic
- In order to use different components, they should be packed according to JavaEE 6 specification, deployed and configured to execute in the container
- Part of the services provided by the container are configurable and allow to change components' behavior according to requirements of the particular deployment (e.g. database access rights)
- Other services are not configurable – e.g. Servlet lifecycle management and database connection pooling

Services Provided by Java™ EE Container (2)

- Among the main services provided by the container are:
  - Declarative security using deployment descriptor or annotations in the code
  - Container-managed transaction demarcation
  - JNDI Lookup services allowing to dynamically allocate objects using symbolic name and thus decoupling components
  - Java™ Remote Connectivity service allowing remote invocation of business methods of another EJB component possibly on another server

Main Advantages of Using Java EE

- More effective management of components life-cycle through reusing existing components
- Remote access to Java EE components and services – distributed enterprise applications – RMI, HTTP/HTTPS
- Standardized and ready to use Java EE services and APIs
- Declarative security, persistence, look up and retrieving objects using symbolic names or new JavaEE 6 Contexts & Dependency Injection (CDI) annotations
- Container managed concurency and transaction demarcation for EJB components

Trayan Iliev
IPT – Intellectual Products & Technologies

Web Services Support

- XML -based web services:
  - Simple Object Access Protocol (SOAP)
    - XML-based envelope
    - XML-based encoding rules
    - XML-based request and response convention
  - Web Services Description Language (WSDL)
  - Universal Description, Discovery and Integration (UDDI) and ebXML Registries integration

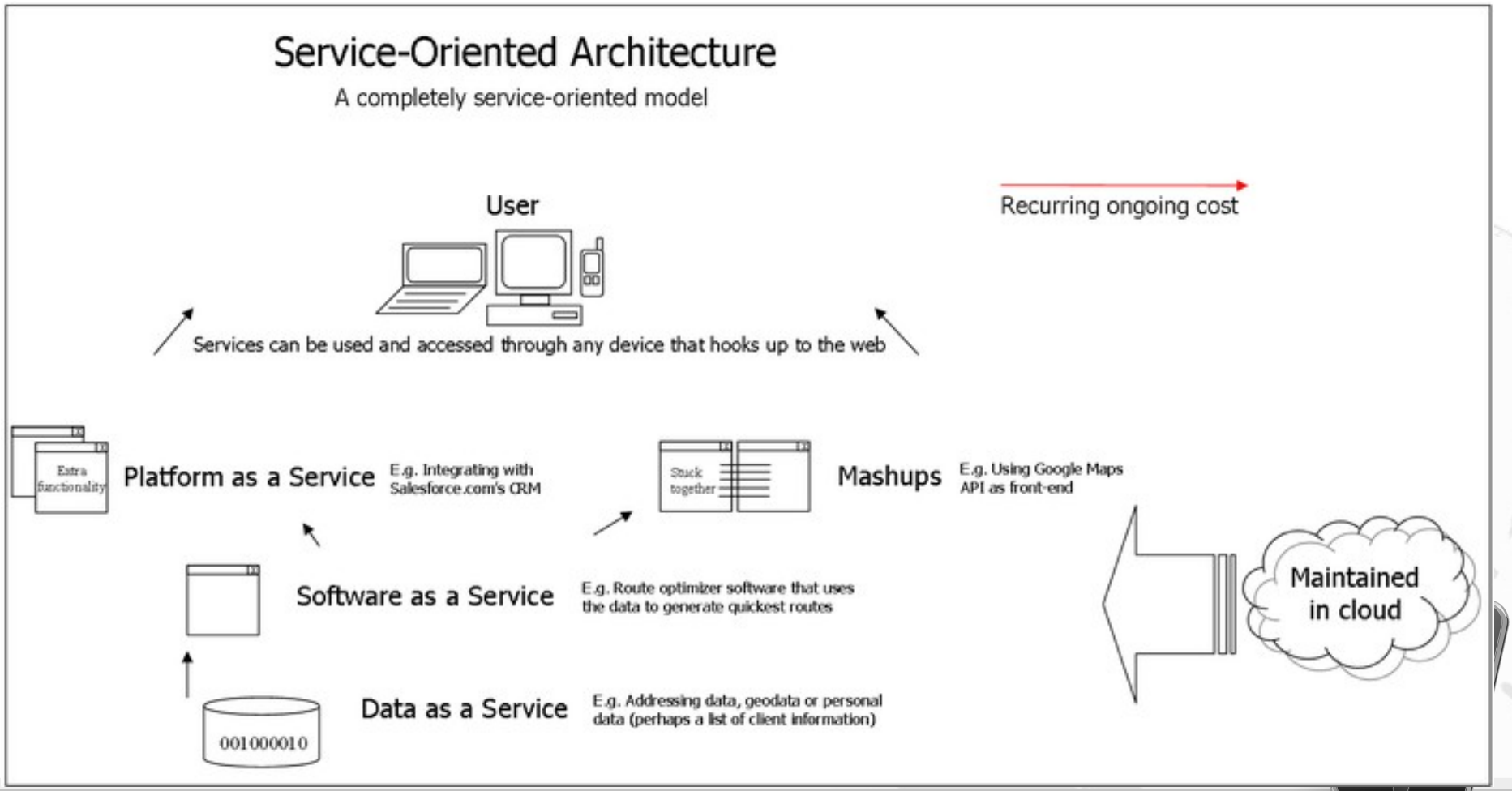
Web Services

Web Services are:

- components for building distributed applications in SOA
- communicate using open protocols
- are self-descriptive and self-content
- can be searched and found using UDDI or ebXML registries

Source: http://en.wikipedia.org/wiki/File:Webservices.png, Author: H. Voormann
License: Creative Commons Attribution 3.0 Unported

Service Oriented Architecture (SOA)



Source: http://en.wikipedia.org/wiki/File:SOA_Detailed_Diagram.png,
Author: JamesLWilliams2010, License: Creative Commons Attribution 3.0 Unported

Trayan Iliev
IPT – Intellectual Products & Technologies

Java™ EE 6 / Java™ SE Standards APIs

According to Java™ EE Specification:

- Web Services
  - Java API for XML Web Services (JAX-WS)
  - Java Architecture for XML Binding (JAXB)
  - SOAP with Attachments API for Java (SAAJ)
  - Java API for XML Registries (JAXR)
  - ~~Java API for XML-based RPC (JAX-RPC)~~
- RESTful Web Services
  - Jersey – RESTful Web Services - JAX-RS

Java EE / Java SE Standards Services (APIs)

According to Java EE Specification (Oracle®/Sun):

- HTTP / HTTPS – Java Servlets, JavaServer Pages (JSP), JavaServer Pages Standard Tag Library (JSTL), JavaServer Faces (JSF), Facelets, Templating & Compositioning, Unified Expression Language (EL)
- Java™ Transaction API (JTA)
- RMI-IIOP
- Java IDL
- JDBC™ API

Trayan Iliev
IPT – Intellectual Products & Technologies

Java EE / Java SE Standards Services (2)

According to Java EE Specification (Oracle®/Sun):

- Java™ Persistence API (JPA)
- Java™ Message Service (JMS)
- Java Naming and Directory Interface™ (JNDI)
- JavaMail™
- JavaBeans™ Activation Framework (JAF)
- XML Processing - Java™ API for XML Processing (JAXP), Java Architecture for XML Binding (JAXB), Streaming API for XML (StAX)

Trayan Iliev
IPT – Intellectual Products & Technologies

Java EE / Java SE Standards Services (3)

According to Java EE Specification (Oracle®/Sun):

- Java EE™ Connector Architecture - contracts:
- Security Services -
  - Java™ Authentication and Authorization Service (JAAS)
  - Java™ Authorization Service Provider Contract for Containers (JACC)
  - Java™ Authentication Service Provider Interface for Containers (JASPIC)
- Management - Java™ Management Extensions (JMX)

Trayan Iliev
IPT – Intellectual Products & Technologies

Novelties in Java™ EE 6

According to Ed Ort's publication from Dec, 2009
http://java.sun.com/developer/technicalArticles/JavaEE/JavaEE6Overview.html
the main goals of Java™ EE 6 are:

- **More flexibility** in combining technologies for different software development niches by defining multiple profiles, first of which is the *Java™ EE 6 Web Profile*
- **Better extensibility** with third-party technologies that are not part of official Java™ EE 6 specification by automatic registration insted of complex .xml descriptions
- **Additional simplification** of the software development process based on POJO (Plain Old Java Object), annotations and dependency injection, recuction of number of interfaces, simplification of JPA object-to-relational mapping

Trayan Iliev
IPT – Intellectual Products & Technologies

Key Improvements in Java™ EE 6 (1)

According to Ed Ort's publication from Dec, 2009
http://java.sun.com/developer/technicalArticles/JavaEE/JavaEE6Overview.html
among the the key improvements in Java™ EE 6 are:

- Java API for RESTful Web Services (JAX-RS)
- Contexts and Dependency Injection for the Java EE Platform (CDI)
- Bean Validation
- Web fragments
- Shared framework pluggability
- Servlet 3.0 (JSR 315) - asynchronous processing, annotations, methods for programmatic authentication, HTTP-only Cookies.
- Simplified creation of web pages with JSF 2.0, Facelets & Templating, Composite components, AJAX support

Trayan Iliev
IPT – Intellectual Products & Technologies

Key Improvements in Java™ EE 6 (2)

According to Ed Ort's publication from Dec, 2009
http://java.sun.com/developer/technicalArticles/JavaEE/JavaEE6Overview.html

- Key improvements in Enterprise Java™ Beans (EJB) 3.1 are:
  - No-interface view
  - Singletons
  - Asynchronous session bean invocation
  - Simplified Packaging
  - EJB Lite
- In Java™ Persistence API JPA (JSR 317) are:
  - Object/Relational mapping improvements
  - Additional keywords in Java Persistence Query Language
  - Brand new Criteria-based programmatic query building API
  - Pessimistic locking support

Trayan Iliev
IPT – Intellectual Products & Technologies

Differences between Full and Web Java™ EE 6 Profiles

- Full EJB 3.1 API including EJB remote interaces, message-driven beans, Web Service EJB endpoints, and EJB Timer Service. (Web profile supports *EJB 3.1 Lite* specification including only *local* stateless session beans, stateful session beans, and singleton session beans)
- Application Client Container
- Java™ Message Service (JMS) resources
- Web services
- Message security
- JavaMail resources
- Connector modules - only outbound communication

Trayan Iliev
IPT – Intellectual Products & Technologies

Key APIs in Java™ EE 6

- javax.ejb.* - EJB
- javax.enterprise.inject.* - CDI
- javax.enterprise.context.* - CDI
- javax.jms.* - JMS
- javax.faces.* - JSF
- javax.faces.component.* - JSF
- javax.mail – Java Mail
- javax.persistence – JPA
- javax.transaction – JTA
- javax.validation – Validation API
- javax.xml.stream - StAX
- javax.resource.* - Java EE Connector Architecture
- javax.jws - JAX-WS
- javax.ws.rs - JAX-RS (RESTful Services)

Java EE Server Support

- GlassFish (https://glassfish.dev.java.net) - supports Java EE 6 Web and Full profiles
- RedHat's JBoss (http://www.jboss.org)- supports Java EE 6 Web profile
- Apache Geronimo (http://geronimo.apache.org/)
- Oracle's WebLogic Suite – part of Fusion Middleware (http://www.oracle.com/us/products/middleware/application-server/index.html)
- IBM's WebSphere (http://www.ibm.com/software/websphere)
- SAP Netweaver (http://www.sdn.sap.com/irj/sdn/nw-products)
- Resin, JOnAS, JEUS, . . .

JSR 299: Contexts and Dependency Injection (1)

- Contexts and Dependency Injection for the Java™ EE Platform starts in June 2006 as JSR 299
- Finalized on December 10th, 2009
- Provides two fundamental services necessary to implement loosely-coupled components with well defined life-cycle:
  - ***Lifecycle Contexts –*** provide well defined life-cycle for managed objects with internal state, in addition to standard contexts (request, session, application, conversation) is possible to define custom ones
  - ***Dependency Injection*** – enables ***typesafe*** component injection (Inversion of control – IoC design principle), allows declarative configuration (choice) of implementation during installation of application (file: beans.xml)

JSR 299: Contexts and Dependency Injection (2)

- Provides modularity for Java EE component architecture taking dependencies into account
- Integrates with ***Unified Expression Language (EL)*** to enable referencing objects in a given context (request, session, application, etc.) directly from ***Facelet*** or ***JSP*** pages
- Supports type-safe ***Interceptors*** for managed objects (beans), separating orthogonal functions (aspects, concerns), and allowing them to be reused together with multiple beans
- ***Decorators*** allow to add functionality to existing beans, compartmentalizing business logic and making it reusable
- ***Events*** allow transparent communication between multiple beans, completely decoupling producers and consumers

JSR 299: Contexts and Dependency Injection (3)

- Adds new web ***Conversation*** context to standard Servlet contexts (***Request***, ***Session*** and ***Application***), allowing to create other custom contexts, if necessary
- Provides full ***Service Provider Interface (SPI)***, which allows easy integration of external libraries with CDI
- Eliminates the need for JNDI lookup based on symbolic names and replaces it with ***typesafe dependency injection***, reducing possibility for runtime type incompatibilities
- Reduces the need for external XML configuration files replacing them with declarative **annotations** in the code, and allowing runtime introspection by development tools

Which Object is Bean According to CDI?

- According to CDI, **Bean** is an object existing in a given context which defines (part of) the state and/or logic of software application
- Java EE class is a bean class if the container can manage the lifecyle of its instances according to CDI contextual model
- Each **bean** has:
  - Non-empty set of **types**
  - Non-empty set of **qualifiers**
  - **Scope** (Request, Session, Application, Conversation, Dependent, etc.)
  - **Implementation**
  - (optional) **Expression Language (EL) name**
  - (optional) **Interceptor bindings**

Which Objects are Managed Beans?

- Objects defined as Managed Beans by some Java EE specification (e.g. JSF, EJB)

  OR

- Fulfills **ALL** the following criteria (no need for special declaration):
  - is not a non-static inner class
  - is not abstract, OR is abstract class annotated as **@Decorator** annotation
  - is not EJB
  - has no-argument constructor, OR constructor annotated with **@Inject** annotation

What Kinds of Objects Can Be Injected with CDI?

- The injection of resources is accomplished by applying ***@Inject*** annotation to fields, constructors, methods, or method parameters
- Types of resources injectable using CDI:
  - All Java classes satisfying the requirements mentioned
  - Session EJB (Stateful, Stateless, Singleton)
  - JPA Persistence Contexts (JPA EntityManager)
  - WebService references
  - References to remote EJB interfaces DataSources, JMS Topics, Queues & Connection Factories
- We can use **qualifier** annotations to distinguish different instances of the same type in a given context – **@Default** is the default qualifier

CDI Defined Scopes

- **@RequestScoped** - particular HTTP request
- **@SessionScoped** - user's active session with the application
- **@ApplicationScoped** - shared scope for all requests of all users of the same application
- **@ConversationScoped** – defined explicitlyprogrammatically) by the JSF application developer – can span over one or more requests with the same **conversation Id (cid)** (long running conversations), cannot cross the session boundaries
- **@Dependent** – default scope in which a new (unique) instances of injected beans are created for each injecting client, with the same scope as the client

Key Annotations in CDI (javax.inject)

- @Inject – Designates injectable constructors, methods, and fields
- @Named – Allows to define EL names of the beans for JSF
- @Qualifier – Designates qualifier annotations
- @Singleton – Designates a type that the injector must instantiate only once

Key Annotations in CDI (javax.enterprise.inject)

- @Alternative – Specifies that a bean is an alternative
- @Any – Every bean has the qualifier @Any, except for the special @New qualified beans
- @Default – Default qualifier type
- @Disposes – Designates the disposed parameter of a disposer method
- @Model = @Named + @RequestScoped (for use in JSF)
- @New – Allows to obtain a new instance of a bean

Key Annotations in CDI (javax.enterprise.inject)

- @Produces – Designates a producer method or field
- @Specializes – Indicates that a bean directly specializes another bean
- @Stereotype – Specifies that an annotation type is a stereotype
- @Typed – Restricts the bean types of a bean

Trayan Iliev
IPT – Intellectual Products & Technologies

Using CDI in Java™ EE 6

- Examples ...

Selection of Alternative Implementations

- **@Alternative –** enables selection of appropriate alternative, implementing the same interface (e.g. for testing purposes) declaratively – in **beans.xml** file:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://java.sun.com/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/beans_1_0.xsd">
    <alternatives>
        <class>mybeans.FormatterTestImpl</class>
    </alternatives>
</beans>
```

Специализация при CDI

- **@Specializes –** allows seamles evolution of code – one more specialized bean can fully substitute in runtime a previous basic bean; the secialized bean must extend the basic bean class, as follows:

```
@Specializes public class BetterTaskImpl extends TaskImpl
{
    .....
}
```

Producer Fields, Methods and Constructors

- **@Produces –** allows to inject objects which are not beans, or which initialization and/or concrete implementation type can vary in runtime – for example:

```
@Produces
   @ChosenFormatter
   @RequestScoped
   public Formatter getFormatter(@New FormatterTestImpl fti,
         @New FormatterImpl fi) {
      switch (chooseFormatter) {
         case 1:   return fti;
         case 2:   return fi;
      }
   }
```

Trayan Iliev
IPT – Intellectual Products & Technologies

Advanced: Events, Interceptors, Decorators

- **Events:** 1) define an event class with one or more qualifiers; 2) annotation **@Observes** + qualifiers on a method parameter allows us to register event listeners; 3) fire event by dependency injection of the interface **Event<T>**, and calling its method **fire(eventPayload)**, where eventPayload is of type **T**
- **Interceptors**: annotations **@InterceptorBinding** and **@Interceptor** + including the interceptor class in the **<interceptors>** element of **beans.xml**, interceptors are reusable
- **Decorators:** annotations **@Decorator** and **@Delegate**, for extending the business functionality of a concrete class through (possibly abstract) decorators

Weld Servlet Injection - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://localhost:8080/ipt-servlet-event/

Google

Weld Servlet... Weld Servlet... Weld Servlet... Weld Servlet... Weld Servlet... Weld Servlet... Weld Servlet... Weld Servlet... Weld Servlet... Weld Servlet... Weld Servlet... Weld Ser...

Add a new product and price:

Product Name:	Thinking In Java	Required
Product Price:	25.00	Required

Submit Reset

CDI_IPT_Iliev.od... ipt-servlet-event ... Inbox - tiliev@ipr... Weld Servlet Inje... Weld Servlet Inje... C:\Business\IPT_... EN 12:47 PM

Weld Servlet Injection - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://localhost:8080/ipt-servlet-event/

Google

Weld Servlet... Weld Servlet... Weld Servlet... Weld Servlet... Weld Servlet... Weld Servlet... Weld Servlet... Weld Servlet... Weld Servlet... Weld Servlet... Weld Servlet... Weld Ser...

Add Product Servlet

Demonstrates Use of CDI

Add a new product and price:

Product Name:	Thinking In Java	Required
Product Price:	25.00	Required

Submit Reset

Product Successfully Added: Thinking In Java

CDI_IPT_Iliev.od... ipt-servlet-event ... Inbox - tiliev@ipr... Weld Servlet Inje... C:\Business\IPT_... IPT-Events.png-1... EN 12:52 PM

Trayan Iliev
IPT – Intellectual Products & Technologies

CDI Events - Define an Event Class

```
package org.iproduct.eshop;

public class NewProductAdded {
    private String productName;

    public NewProductAdded(String productName) {
        this.productName = productName;
    }
}
```

CDI Events – @Inject Event<T>, fire(eventData)

```
@Inject @Any
 Event<NewProductAdded> newProductEvent;
 private boolean empty = true;
 public void addProduct() {
     if ((pData.getProductName() != null
             && pData.getProductName().length() > 0)
             && (pData.getProductPrice() > 0)) {
         empty = false;
         // Fire an event
         newProductEvent.fire(
             new NewProductAdded(pData.getProductName()));
     }
 }
```

CDI Events – Event Listeners: @Observes

```
public void procuctAdded(
    @Observes NewProductAdded event) {
    System.out.println("----> ProductAddedEvent catched: " +
          event);
}
```

Trayan Iliev
IPT – Intellectual Products & Technologies

Interceptors – @InterceptorBinding annotation

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

Interceptors – @Interceptor

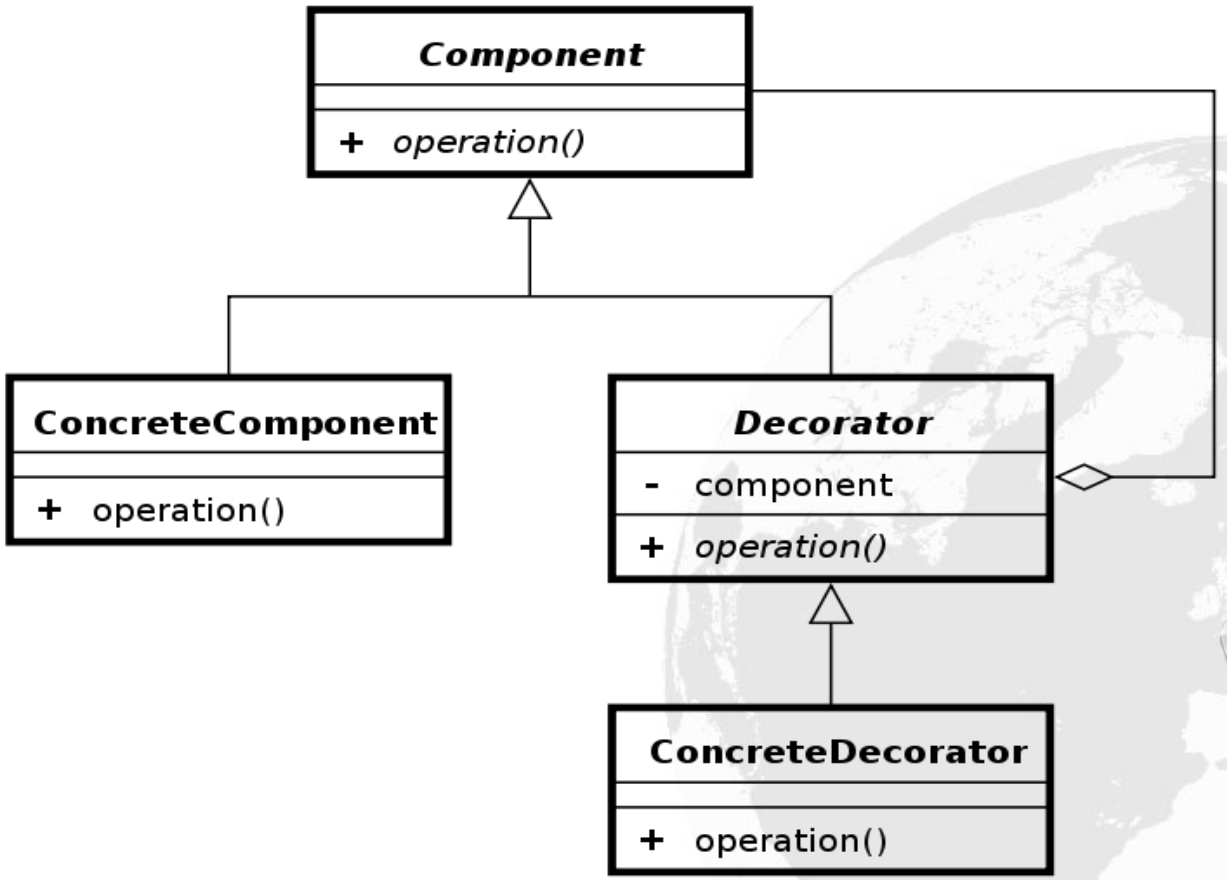
```
@Logged @Interceptor
public class LoggedInterceptor implements Serializable {
  private Logger logger =
      Logger.getLogger("org.iproduct.eshop.interceptor");
  @AroundInvoke
  public Object logMethodEntry(InvocationContext invContext)
          throws Exception {
      logger.info("Entering method: "
 + invocationContext.getMethod().getName() + " in class "
 + invocationContext.getMethod().getDeclaringClass().getName());
      return invocationContext.proceed();
  }
}
```

Interceptors – <interceptors> in beans.xml

```
<beans xmlns="http://java.sun.com/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
         http://java.sun.com/xml/ns/javaee/beans_1_0.xsd">
    <interceptors>
      <class>billpayment.interceptor.LoggedInterceptor</class>
    </interceptors>
</beans>
```

```
@Logged
@Named @ApplicationScoped @Default
public class ProductController implements Serializable {
   ...
```

Reusable Design Pattern: Decorator



Source: http://en.wikipedia.org/wiki/File:Decorator_UML_class_diagram.svg
License: public domain

Trayan Iliev
IPT – Intellectual Products & Technologies

Decorators – @Decorator and @Delagate

```
@Decorator
public abstract class FormatterDecorator implements Formatter {
  @Inject
  @Delegate
  @Any
  Formatter delegate;
  public static final Logger logger =
        Logger.getLogger("org.iproduct.eshop.decorators");
  @Override
  public String formatString(String s) {
     logger.log(Level.INFO, "Entering decorator method.");
     return delegate.formatString(s).toUpperCase();
  }
}
```

Trayan Iliev
IPT – Intellectual Products & Technologies

Decorators – <decorators> in beans.xml

```
<beans xmlns="http://java.sun.com/xml/ns/javaee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/beans_1_0.xsd">
      <decorators>
            <class>decorators.CoderDecorator</class>
      </decorators>
</beans>
```

References

- JSR 244: JavaTM Platform, Enterprise Edition 5 (Java EE 5) Specification – http://jcp.org/en/jsr/detail?id=244
- Java EE 6 Tutorial – http://java.sun.com/javaee/6/docs/tutorial/doc/
- Introducing the Java EE 6 Platform – http://java.sun.com/developer/technicalArticles/JavaEE/JavaEE6Overview.html
- JSR 299: Contexts and Dependency Injection for the JavaTM EE platform – http://jcp.org/en/jsr/detail?id=299
- JSR 330: Dependency Injection for Java – http://jcp.org/en/jsr/detail?id=330
- Sang Shin's Java Passion educational web site – http://www.javapassion.com/

Thanks for Your Attention!

Questions?