JEE Basics

Udemy Course by Tim Buchalka

Table of Contents

[1. The Theory of Java EE 3](#_Toc89301713)

[1.1. What is it? 3](#_Toc89301714)

[1.2. What is an Application Server? 3](#_Toc89301715)

[1.3. What is a JSR? 3](#_Toc89301716)

[1.4. Reference Implementation 3](#_Toc89301717)

[1.5. Jakarta EE 4](#_Toc89301718)

[1.6. Java EE and Spring Framework 4](#_Toc89301719)

[2. Software to install 4](#_Toc89301720)

[2.1. Helloworld 4](#_Toc89301721)

[3. Todo App 4](#_Toc89301722)

[4. What is JEE so far? 6](#_Toc89301723)

[5. Contexts and Dependency Injection (CDI 2.0) 6](#_Toc89301724)

[5.1. Features 6](#_Toc89301725)

[5.2. Contextual Instance 7](#_Toc89301726)

[5.3. CDI Lifecycle callback - PostConstruct 8](#_Toc89301727)

[5.4. Get your hands dirty 9](#_Toc89301728)

[5.5. Managed Beans and Bean Types 9](#_Toc89301729)

[5.5.1. Qualifiers 10](#_Toc89301730)

[5.6. CDI Stereotypes 11](#_Toc89301731)

[5.7. CDI Scopes and Contexts 12](#_Toc89301732)

[5.7.1. Session Scope 12](#_Toc89301733)

[5.7.2. Application and Conversation Scope 13](#_Toc89301734)

[5.8. Producers in CDI 13](#_Toc89301735)

[5.8.1. CDI Producers - Qualifying Beans 14](#_Toc89301736)

[5.9. CDI Interceptors 15](#_Toc89301737)

[5.10. Events 16](#_Toc89301738)

[5.10.1. Interface Event<T> 16](#_Toc89301739)

[o Method Summary 17](#_Toc89301740)

[o Method Detail 18](#_Toc89301741)

[5.11. Event Interface 20](#_Toc89301742)

# The Theory of Java EE

# What is it?

Collection of abstracs specifications that offer a solution for solving common challenges in the Developer's world. An annotation driven platform (Framework, like Java Spring).

* Abstract Specs
* Commonly Faced Challenges - Persistence, web services, transactions, security, lose coupling, etc.
* Focus is on abstract (Developers can focus on solving the challenge at hand, instead on building usual libraries, that are already in the JEE set)

# What is an Application Server?

* Concrete implementation - application server
* Abstract from detailed implementation
* javax.\* package

Examples of Java EE App Servers

* Payara server (Glassfish) - <https://payara.fish>
* IBM OpenLiberty - <https://openliberty.io>
* JBOSS Wildfly - <http://www.wildfly.org/>

# What is a JSR?

Java Specification Request

* Formal request to Java Community Process (JCP) (a body that standardizes APIs on the Java technology platform.
* (for) New proposal, enhancement to existing APIs eg CDI 1 -> CDI 2 (to the java platform)
* Used to group APIs into silos

Examples:

* <https://jcp.org/en/home/index>
* <https://jcp.org/en/jsr/platform>
* Link for the JEE Requests: <https://jcp.org/en/jsr/platform?listBy=3&listByType=platform>

# Reference Implementation

* Concrete realization of the abstract JSR
* Eg JAX-RS reference implementation - Jersez
* Java EE itself is a JSR!
* Java EE 8 is JSR 366 - RI Glassfish 5

Application server redefinition: Collection of the various reference implementations for the JSRs.

# Jakarta EE

(Bord when Oracle open up the development of JEE to the developers community)

* Java EE going forward
* Hosted by Eclipse Foundation
* https://jakarta.ee

# Java EE and Spring Framework

* Java EE influenced by Spring Framework
* Spring boot influenced by Java EE
* Both good platforms - use what works best and solves your problem
* No flame wars - quite pointless

Spring Framework (more simplified) both frameworks influence each other.

# Software to install

* Apache maven: <https://www.baeldung.com/install-maven-on-windows-linux-mac>
* Insomnia: <https://docs.insomnia.rest/insomnia/install>
* apache netbeans: <https://netbeans.apache.org/>
* git: <https://git-scm.com/downloads>
* payara: <https://www.payara.fish/>

# Helloworld

Dynamic Web App (Netbean Project hello-javaee8)

Deploy a web app using the payara.jar from the command line:

java -jar payara.jar --deploy /Users/lluis.carrasco.martinez-cic@ibm.com/NetBeansProjects/hello-javaee8/target/hello-javaee8.war --port 8080

See the app in the browser:

http://localhost:8080/hello-javaee8/resources/ping

# Todo App

Open NetBeans

* New Project
  + Maven Project from archetype
  + javaee8-essentials-archetype
  + Name the project "hello-todo"
  + Group Id: academy.learnprogramming
  + Package: academy.learnprogramming
  + Click on "Finish".
* In the created Project, in the "Project Files/pom.xml" file are listed the dependencies.
* To import the java library of a class in Netbeans press "Alt + enter" keys
* To automatically generate the Getters and Setters, go to the Menu:
  + Source
  + Insert Code
  + Getters and Setters
* The persistence.xml information is located in "Other Sources/src/main/resources/META-INF/"

Text

Description automatically generated

* Set tomcat: <https://www.dariawan.com/tutorials/java/how-to-add-apache-tomcat-server-in-netbeans-10/>
* Open a terminal in the project folder (or a command line) and prompt: mvn package (Maven will create our Web Archive of the application).
  + a <project-name>.war file will be created
* Using payara, like in the example before, we will deploy the application:
  + From the terminal in the folder where payara resides, prompt the next line to deploy the application: java -jar payara.jar --deploy [/Users/lluis.carrasco.martinez-cic@ibm.com/NetBeansProjects/hello-todo/target/hello-todo.war](mailto:/Users/lluis.carrasco.martinez-cic@ibm.com/NetBeansProjects/hello-todo/target/hello-todo.war) --port 8080
* Open the Insomnia Client to test.
* Prompt the following line in the folder where our payara.jar file resides to create an UberJar bundle: java -jar payara.jar --deploy [/Users/lluis.carrasco.martinez-cic@ibm.com/NetBeansProjects/hello-todo/target/hello-todo.war](mailto:/Users/lluis.carrasco.martinez-cic@ibm.com/NetBeansProjects/hello-todo/target/hello-todo.war) --port 8088 --outputUberJar helloTodo.jar
* Prompt to run the application: java -jar helloTodo.jar
* Another way to package the application is by using the concept of **maven profiles**. He is using the **payara-micro** package, however, he does not explains how to get it into the project, neither he passed the **hello-todo** maven configuration file. (Prompt: mvn package payara-micro:start)

# What is JEE so far?

Has 3 APIs that we should master:

* JPA (Java Persistence API, responsible for
  + Storing information in the relational DB
  + Retrieving information from the relational DB
* CDI (Context and Dependency Injection)
  + Dependency Injection API on the JEE platform
  + To create highly coupled applications
  + Act as a bridge between the JPA and the JAX-RS
* Jax-RS
  + To expose the resources over the HTTP protocol
  + Standardized REST resources for the end points
  + Expose the resources of the Web App to the world (in the internet)

# Contexts and Dependency Injection (CDI 2.0)

What is Dependency Injection: Specific form of versional control (software where individual components have the dependencies supply to them instead of creating them themselves). The dependencies are declared in the entity manager. We externalize the creation of dependencies and objects in the project to a CDI container, as the project gets larger, it becomes easier to maintain.

# Features

**Typesafe Dependency Injection**: We can declare the dependencies in types. Feature offered by the Java EE CDI. (Type errors can be know at compile time.

**Lifecycle Contexts**: The components we declare dependencies on have a life cycle, (is it a Session based object, a Request based object, ...?) We manage the life cycles with the annotations provided on the java EE Platform.

**Interceptors**: Intercept requests to methods (Crosscutting, ...)

Way to developed highly decoupled applications. One component can send data to another component, just by listening to events. So one component can send events, while another is listening.

**Service Provider Interface**: Self provider Interface (APIs)

**Bean Discovery Mode:** The way how the Dependency Injection Runtime analyzes and discovers the beans (beans.xml). Process the CDI uses to discover beans that must be managed:

<beans bean-discovery-mode="all"> (if it is annotated, only the annotated (with Annotations) beans will be discovered, or if we set it to "none", we just disable the discovery mode)

</beans>

**What is in the CDI a container**:

* a black box, that takes all our java classes and turn them in something we can use it in our application, like a factory, get an input and generates and output. It is the application that makes almost of the magic for us.
* It manages the life cycle of the beans
* Manages the interacting context between beans.

**What is a Bean:**

* A template that a developer creates (A java Class, to which we give an annotation)
* Most of the time in JEE we create beans that will be instantiated (contextual instance).

# Contextual Instance

* An instance of a bean that is created and manage by the CDI container.
  + @Inject
  + private RequestScope reqeustScope; // for example

**CDI Injection Point - Field**

* An Injection point is the point is the point at which the CDI container inject the point for your. A class where we find the
  + @Inject
  + private MyClass myClass;
* This injection happens inside a Java Class, e.g see image below

Graphical user interface, text, application, chat or text message, email

Description automatically generated

This initializes our beans.

**CDI Injection Point - Constructor**

* We are calling for an injection of a class. In this example, the CDI will create a contextual instance of the DependentScope class, that has been injected previously. All the parameters in the constructor are eligible beans, that have been injected.

Graphical user interface, text

Description automatically generated

**CDI Injection Point - Method**

Graphical user interface, text

Description automatically generated

We are passing as a parameter a contextual instance of a bean (a SessionScope), to instantiate our own SessionScope. We have to annotate the method with the @Inject Annotation.

# CDI Lifecycle callback - PostConstruct

A point in the life cycle of a bean that a CDI container gives the opportunity to hook into. 2 callbacks

* @PostConstruct >> The method will be called after the bean has been created (the bean is ready for use, but has not been used)

A picture containing text

Description automatically generated

Any initialized bean can be used here without becoming any NullPointerException. In our example we just printed some logged messages.

* @PreDestroy >> a life cycle call back invoke just before the bean is destroyed (called for garbage collection).

Text

Description automatically generated with low confidence

# Get your hands dirty

* Deploy the project "javaee-fundamentals" using: mvn package in a terminal opened in the project folder.
* Open a terminal in the project and prompt the following line to deployed the application: java -jar payara.jar --deploy [/Users/lluis.carrasco.martinez-cic@ibm.com/NetBeansProjects/javaee-fundamentals/target/cdi.war](mailto:/Users/lluis.carrasco.martinez-cic@ibm.com/NetBeansProjects/javaee-fundamentals/target/cdi.war) --port 8089
* Go in your brower to <http://localhost:8089/cdi/> to see the application.

# Managed Beans and Bean Types

A managed bean is any bean eligible for managing by CDI

A bean type: TypeSafe Resolution (concrete java Types) Concrete type of a bean. Using Type Bean Resolution means we can inject by type (Inheritage). In the project javaee-fundamentals, if a class implements Salute (the interface), then it is of type "Salute". So If 2 classes are of type "Salute", we can indicate the specific type of the instantiation by Annotatitions:

* private class Police implements Salute {}
* In the corresponding Bean:
  + @Inject
  + @Police // to indicate the specific type also called qualifier for it specifies the type.
  + private Salute policeSalute;

# Qualifiers

Example of a Qualifier:

Graphical user interface, text, application

Description automatically generated

We have provided this class with the Annotation Police, to avoid ambiguities, as we read in the highlighted line (see picture below)

Graphical user interface, text, application, email

Description automatically generated

Qualifier: to distinguish the beans, mechanism for distinguising bean types and avoiding ambiguities.

Making the qualification by using the class ServiceMan (see the project **javaee-fundamentals** for more, file **QualifiersWithValueBean.java**).

Graphical user interface, text, application, email

Description automatically generated

# CDI Stereotypes

Collection of Annotations group together as one

Example of a stereotype

Graphical user interface, text, application

Description automatically generated

This stereotype englobes the following Annotations:

Graphical user interface, text, application, email

Description automatically generated

Anywhere where we want to use @RequestScoped and @Name, we can use @Web (saving code and time)

Otherwise we will have to give the 2 Annotations to both classes: Police and Soldier:

Graphical user interface, text, application, email

Description automatically generated

# CDI Scopes and Contexts

**Scope**: a way to tell the container to associate specific contextual instance with a given context.

In the next image, the @RequestScoped is the annotation thats sets the Scope for this bean.

Graphical user interface, text, application

Description automatically generated

Any bean without a defined scope falls into the **@Dependent** scope, meaning they will take the context of the bean where they are being injected, because it is called a Dependence Scope Bean.

**Context**: refers a valid environment where a well-defined lifecycle and life time of a bean can reside (be created and destroyed).

CDI container takes care of the creation-life cycle - destroying of the context.

# Session Scope

A bean whose contextual scope will be bound to an HTTP Session.

Every HTTP session refers to a long communication session between the client and the server.

The beans context is bound to a Session, therefore the lifecycle is bound to the http session.

Graphical user interface, text, application, email

Description automatically generated

Context Instances are only created when they are requested, upon request (generally).

Passivating: putting the bean into an hibernate state when it is not being used, and it will be activated when it is needed (CDI-API).

# Application and Conversation Scope

The **ApplicationScope** binds contextual instances of the bean to the lifecycle of the application itself.

Graphical user interface, text, application, email

Description automatically generated

As only one instantiation will be done, it is use normally as a Singleton Pattern.

**ConversationScope**: used with the Java Server Faces API, it is manually manage by the developer. It is a bean that transverses HTTP request. It is available if needed (written in the program). The instance will be created, and it will be active until the developer declares the end of it (features updates, etc.).

# Producers in CDI

API construct that enables the developer to turn classes into CDI injectable beans.

**Method producer**

Graphical user interface, text, application, email

Description automatically generated

In the code example above, the method is annotated with the Annotation **@Produces**, turning the method into a CDI producer method, the method must then return something, must have a return type. The parameters become injection points, the CDI container will take care of injecting the parameters. Therefore, the parameters must be beans (injectable objects), so the CDI container can inject them.

**Field producer:**

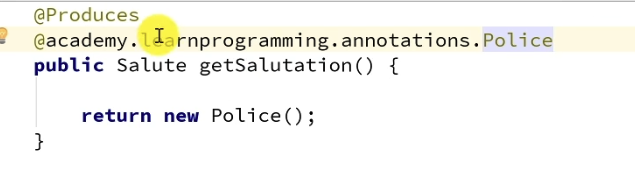
Graphical user interface, text, application

Description automatically generated

Hence the Annotations @Producers and @PersistenceContext, entityManager is now a producer field, and can be injected.

# CDI Producers - Qualifying Beans

Used to clarify ambiguities.



If we qualify the method with @academy... , we avoid ambiguity, for Police implements Salute. Now the CDI container knows it is type Police.

Text

Description automatically generated

In this example, the method **dispose()** cleans the List created by **getLuckyDish()**.

# CDI Interceptors

Intercept a CDI API construct, a way to intercept calls to methods or entire methods into a given class, and do some kind of work (security, login, ...) and decide if it has to procede or not.

Graphical user interface, text, application, email

Description automatically generated

* An Interceptor binding code must be written (Annotation **@InterceptorBinding**). It is the code that triggers an interceptor.
* Once an Interceptor is triggered, the piece of Code of the Interceptor will run. See the image below to check an example of Interceptor Binding Code from the @interface above.

Graphical user interface, text, application

Description automatically generated

The picture above, is the code that will run if the Interceptor is triggered. The method with the annotation **@AroundInvoke**, is the method that will be invoked by the interceptor. The parameter of the method is from type **InvocationContext**.

The Interceptor is activated thanks to the **@Priority()** Annotation. In the example **@Priority(Interceptor.Priority.APPLICATION)**

Name the Interceptors with meaningful names (in the example, as it is used for a log in the Annotation used is @Logged)

# Events

Any user action, or subsequent occurrence that happens in the web app due to the user activity (Changes in the Web application UI generated by the User actions (Check if there can be Events that can be generated without the user taking part on it)).

The CDI Events API Constructs help us to build reactive Web Applications.

The most important API to manage the Events in Java EE:

**Interface Event** (<https://javaee.github.io/javaee-spec/javadocs>)

public interface Event<T>. (The T is the Pay Load).

# Interface Event<T>

* Type Parameters:

T - the type of the event object

public interface Event<T>

Allows the application to fire events of a particular type.

Beans fire events via an instance of the Event interface, which may be injected:

@Inject

@Any

Event<LoggedInEvent> loggedInEvent;

The fire() method accepts an event object:

public void login() {

...

loggedInEvent.fire( new LoggedInEvent(user) );

}

Any combination of qualifiers may be specified at the injection point:

@Inject

@Admin

Event<LoggedInEvent> adminLoggedInEvent;

Or, the [@Any](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/inject/Any.html" \o "annotation in javax.enterprise.inject) qualifier may be used, allowing the application to specify qualifiers dynamically:

@Inject

@Any

Event<LoggedInEvent> loggedInEvent;

For an injected Event:

* + the specified type is the type parameter specified at the injection point, and
  + the specified qualifiers are the qualifiers specified at the injection point.

Events may also be fired asynchronously with [fireAsync(Object)](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/Event.html" \l "fireAsync-U-) and [fireAsync(Object, NotificationOptions)](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/Event.html" \l "fireAsync-U-javax.enterprise.event.NotificationOptions-) methods

Author:

Gavin King, Pete Muir, David Allen, Antoine Sabot-Durand

### Method Summary

|  |  |
| --- | --- |
| All Methods[Instance Methods](javascript:show(2);)[Abstract Methods](javascript:show(4);) | |
| **Modifier and Type** | **Method and Description** |
| void | [fire](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/Event.html#fire-T-)([T](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/Event.html" \o "type parameter in Event) event)  Fires an event with the specified qualifiers and notifies observers. |
| <U extends [T](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/Event.html" \o "type parameter in Event)> [CompletionStage](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletionStage.html?is-external=true)<U> | [fireAsync](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/Event.html#fireAsync-U-)(U event)  Fires an event asynchronously with the specified qualifiers and notifies asynchronous observers. |
| <U extends [T](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/Event.html" \o "type parameter in Event)> [CompletionStage](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletionStage.html?is-external=true)<U> | [fireAsync](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/Event.html#fireAsync-U-javax.enterprise.event.NotificationOptions-)(U event, [NotificationOptions](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/NotificationOptions.html" \o "interface in javax.enterprise.event) options)  Fires an event asynchronously with the specified qualifiers and notifies asynchronous observers. |
| [Event](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/Event.html)<[T](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/Event.html" \o "type parameter in Event)> | [select](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/Event.html#select-java.lang.annotation.Annotation...-)([Annotation](http://docs.oracle.com/javase/8/docs/api/java/lang/annotation/Annotation.html?is-external=true" \o "class or interface in java.lang.annotation)... qualifiers)  Obtains a child Event for the given additional required qualifiers. |
| <U extends [T](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/Event.html" \o "type parameter in Event)> [Event](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/Event.html)<U> | [select](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/Event.html#select-java.lang.Class-java.lang.annotation.Annotation...-)([Class](http://docs.oracle.com/javase/8/docs/api/java/lang/Class.html?is-external=true" \o "class or interface in java.lang)<U> subtype, [Annotation](http://docs.oracle.com/javase/8/docs/api/java/lang/annotation/Annotation.html?is-external=true" \o "class or interface in java.lang.annotation)... qualifiers)  Obtains a child Event for the given required type and additional required qualifiers. |
| <U extends [T](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/Event.html" \o "type parameter in Event)> [Event](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/Event.html)<U> | [select](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/Event.html#select-javax.enterprise.util.TypeLiteral-java.lang.annotation.Annotation...-)([TypeLiteral](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/util/TypeLiteral.html" \o "class in javax.enterprise.util)<U> subtype, [Annotation](http://docs.oracle.com/javase/8/docs/api/java/lang/annotation/Annotation.html?is-external=true" \o "class or interface in java.lang.annotation)... qualifiers)  Obtains a child Event for the given required type and additional required qualifiers. |

### Method Detail

#### fire

void fire([T](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/Event.html" \o "type parameter in Event) event)

Fires an event with the specified qualifiers and notifies observers.

Parameters:

event - the event object

Throws:

[IllegalArgumentException](http://docs.oracle.com/javase/8/docs/api/java/lang/IllegalArgumentException.html?is-external=true) - if the runtime type of the event object contains a type variable

[ObserverException](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/ObserverException.html) - if a notified observer throws a checked exception, it will be wrapped and rethrown as an (unchecked) [ObserverException](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/ObserverException.html" \o "class in javax.enterprise.event)

#### fireAsync

<U extends [T](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/Event.html" \o "type parameter in Event)> [CompletionStage](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletionStage.html?is-external=true" \o "class or interface in java.util.concurrent)<U> fireAsync(U event)

Fires an event asynchronously with the specified qualifiers and notifies asynchronous observers.

Parameters:

event - the event object

Returns:

a [CompletionStage](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletionStage.html?is-external=true" \o "class or interface in java.util.concurrent) allowing further pipeline composition on the asynchronous operation. Default asynchronous execution facility is container specific. If any observer notified by this event throws an exception then the resulting CompletionStage is completed exceptionally with [CompletionException](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletionException.html?is-external=true" \o "class or interface in java.util.concurrent) that wraps all the exceptions raised by observers as suppressed exception. If no exception is thrown by observers then the resulting CompletionStage is completed normally with the event payload.

Throws:

[IllegalArgumentException](http://docs.oracle.com/javase/8/docs/api/java/lang/IllegalArgumentException.html?is-external=true) - if the runtime type of the event object contains a type variable

Since:

2.0

#### fireAsync

* + - <U extends [T](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/Event.html" \o "type parameter in Event)> [CompletionStage](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletionStage.html?is-external=true" \o "class or interface in java.util.concurrent)<U> fireAsync(U event,

[NotificationOptions](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/NotificationOptions.html) options)

Fires an event asynchronously with the specified qualifiers and notifies asynchronous observers. A custom [Executor](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executor.html?is-external=true" \o "class or interface in java.util.concurrent) will be used to make asynchronous calls

Parameters:

event - the event object

options - the notification options

Returns:

a [CompletionStage](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletionStage.html?is-external=true" \o "class or interface in java.util.concurrent) allowing further pipeline composition on the asynchronous operation. Default asynchronous execution facility is container specific. If any observer notified by this event throws an exception then the resulting CompletionStage is completed exceptionally with [CompletionException](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletionException.html?is-external=true" \o "class or interface in java.util.concurrent) that wraps all the exceptions raised by observers as suppressed exception. If no exception is thrown by observers then the resulting CompletionStage is completed normally with the event payload.

Throws:

[IllegalArgumentException](http://docs.oracle.com/javase/8/docs/api/java/lang/IllegalArgumentException.html?is-external=true) - if the runtime type of the event object contains a type variable

Since:

2.0

#### select

[Event](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/Event.html)<[T](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/Event.html" \o "type parameter in Event)> select([Annotation](http://docs.oracle.com/javase/8/docs/api/java/lang/annotation/Annotation.html?is-external=true" \o "class or interface in java.lang.annotation)... qualifiers)

Obtains a child Event for the given additional required qualifiers.

Parameters:

qualifiers - the additional specified qualifiers

Returns:

the child Event

Throws:

[IllegalArgumentException](http://docs.oracle.com/javase/8/docs/api/java/lang/IllegalArgumentException.html?is-external=true) - if passed two instances of the same non repeating qualifier type, or an instance of an annotation that is not a qualifier type

#### select

* + - <U extends [T](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/Event.html" \o "type parameter in Event)> [Event](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/Event.html" \o "interface in javax.enterprise.event)<U> select([Class](http://docs.oracle.com/javase/8/docs/api/java/lang/Class.html?is-external=true" \o "class or interface in java.lang)<U> subtype,

[Annotation](http://docs.oracle.com/javase/8/docs/api/java/lang/annotation/Annotation.html?is-external=true)... qualifiers)

Obtains a child Event for the given required type and additional required qualifiers.

Type Parameters:

U - the specified type

Parameters:

subtype - a [Class](http://docs.oracle.com/javase/8/docs/api/java/lang/Class.html?is-external=true" \o "class or interface in java.lang) representing the specified type

qualifiers - the additional specified qualifiers

Returns:

the child Event

Throws:

[IllegalArgumentException](http://docs.oracle.com/javase/8/docs/api/java/lang/IllegalArgumentException.html?is-external=true) - if passed two instances of the same non repeating qualifier type, or an instance of an annotation that is not a qualifier type

#### select

* + - <U extends [T](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/Event.html" \o "type parameter in Event)> [Event](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/event/Event.html" \o "interface in javax.enterprise.event)<U> select([TypeLiteral](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/util/TypeLiteral.html" \o "class in javax.enterprise.util)<U> subtype,

[Annotation](http://docs.oracle.com/javase/8/docs/api/java/lang/annotation/Annotation.html?is-external=true)... qualifiers)

Obtains a child Event for the given required type and additional required qualifiers.

Type Parameters:

U - the specified type

Parameters:

subtype - a [TypeLiteral](https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/util/TypeLiteral.html" \o "class in javax.enterprise.util) representing the specified type

qualifiers - the additional specified qualifiers

Returns:

the child Event

Throws:

[IllegalArgumentException](http://docs.oracle.com/javase/8/docs/api/java/lang/IllegalArgumentException.html?is-external=true) - if passed two instances of the same non repeating qualifier type, or an instance of an annotation that is not a qualifier type

# Event Interface

* Simple Events: any Observer that is observing for this particular event will be invoke by the container automatically when we fire the particular events coded in the code.
* Qualifying Events: When it is only wanted that one Observer is triggered, or activated, a qualifier is used, to indicate the CDI, to call the observer that has the same qualifier as the Event:
  + @Qualifier PopularStand is created

Graphical user interface, text, application, email

Description automatically generated

* + We annotate it in the EventBean.java by the injection with the created Qualifier

Text

Description automatically generated with medium confidence

* + The Observer is also with the particular qualifier, so when the CDI container, know that for the particular event, only this Observer has to be activated. It allows us to specify what observer to what event matches.

Graphical user interface, text, application

Description automatically generated

* Conditional Observers: the **notifyObserver** is what allow us to add the condition

Text

Description automatically generated

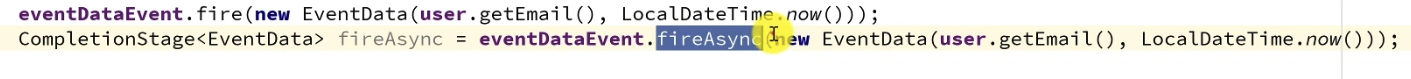
The conditions are added inside the @Observes annotation.

* Async Events: @ObservesAsync

Graphical user interface, text, application

Description automatically generated

Inside we need to be fired by an asynchronous event:



The event uses the fireAsync() to fire an asynchronous event.

* Prioritizing Observer Method Invocation: We can order the sequence in which the Observers are going to be invoked.

Graphical user interface, text, application

Description automatically generated

We give priority with the Anotation @Priority(n) where n is an integer, the lower, the higher the priority.

The corrected code below, using the API

Graphical user interface, text, application, chat or text message

Description automatically generated