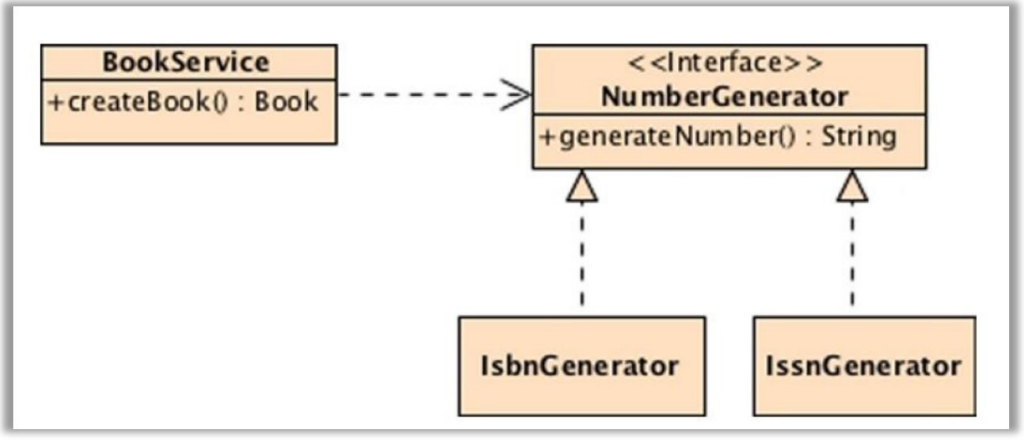
Context and Dependency Injection (CDI)

Dependency Injection

- في البرمجة الشيئية مجموعة من objects تتعامل مع بعضها البعض وعندما يعتمد object على object اخر ويحصل عليه من الخارج من غير ان يقوم بتكوينه مباشرة هذا يعرف ب DI .

- Class diagram التالي يبين BookService class تعتمد على NumberGenerator interface للحصول على book number .



Dependency Injection

يمكن ان تحصلBookService class على NumberGenerator بأحد الطريقتين.

Using New Keyword

```
public class BookService {

  private NumberGenerator numberGenerator;

  public BookService() {
    this.numberGenerator = new IsbnGenerator();
  }

  public Book createBook(String title, Float price, String description) {
    Book book = new Book(title, price, description);
    book.setIsbn(numberGenerator.generateNumber());
    return book;
  }
}
```

A BookService POJO Creating Dependencies Using the New Keyword

Construction by hand

```
public class BookService {

  private NumberGenerator numberGenerator;

  public BookService(NumberGenerator numberGenerator) {
    this.numberGenerator = numberGenerator;
  }

  public Book createBook(String title, Float price, String description) {
    Book book = new Book(title, price, description);
    book.setIsbn(numberGenerator.generateNumber());
    return book;
  }
}
```

A BookService POJO Choosing Dependencies Using the Constructor

```
BookService bookService = new BookService(new IsbnGenerator())
```

CDI

- يعتبر standard dependency injection framework الذي تقدمه Java EE .
- يسمح بإدارة دورة حياة Objects من خلال domain specific lifecycle contexts وعمل Inject لها في Objects الأخرى بما يعرف type-safe way .

Writing CDI Bean

- هي POJO class لاتعمل extend لاية class أخرى.
- يمكن ان تكون اي جافا class تحتوي على business logic .
- يمكن استدعائها مباشرة من خلال Inject من أي برنامج جافا او يتم استدعائها من خلال EL في صفحات JSF .

CDI Bean

- التالي يبين CDI Bean.

```java
@Named(value = "loginBean")
@SessionScoped
public class LoginBean implements Serializable {

    private String username;
    private String password;

    public LoginBean() {
    }
    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }

        public String submit(){

           if (!password.equals("12345")) {
            FacesContext context = FacesContext.getCurrentInstance();
            FacesMessage message = new FacesMessage("Authentication failed");
            context.addMessage(null, message);
            return null;
        } else {
            return "main";
        }
    }
}
```

Anatomy of a CDI Bean

- أي Java Class تحقق الشروط التالية تعتبر CDI Bean .
- ليست non-static inner class .
- تكون concrete class او تم عمل annotation لها باستخدام @Decorator.
- تحتوي على default constructor with no parameters

@Inject

- عند استخدام Java EE لا توجد حاجة لاستخدام construct by hand لانه container ستقوم بذلك بالنيابة عنك.
- مع CDI يمكن القيام بذلك باستخدام @Inject annotation كما هو مبين بالاسفل.

```
public class BookService {

  @Inject
  private NumberGenerator numberGenerator;

  public Book createBook(String title, Float price, String description) {
    Book book = new Book(title, price, description);
    book.setIsbn(numberGenerator.generateNumber());
    return book;
  }
}
```

```
public class IsbnGenerator implements NumberGenerator {

  public String generateNumber() {
    return "13-84356-" + Math.abs(new Random().nextInt());
  }
}
```

Injection Points

- يمكن استخدام @Inject في ثلاث أماكن وهي property, setter, constructor .

Property injection point

```
@Inject
private NumberGenerator numberGenerator;
```

Constructor injection point

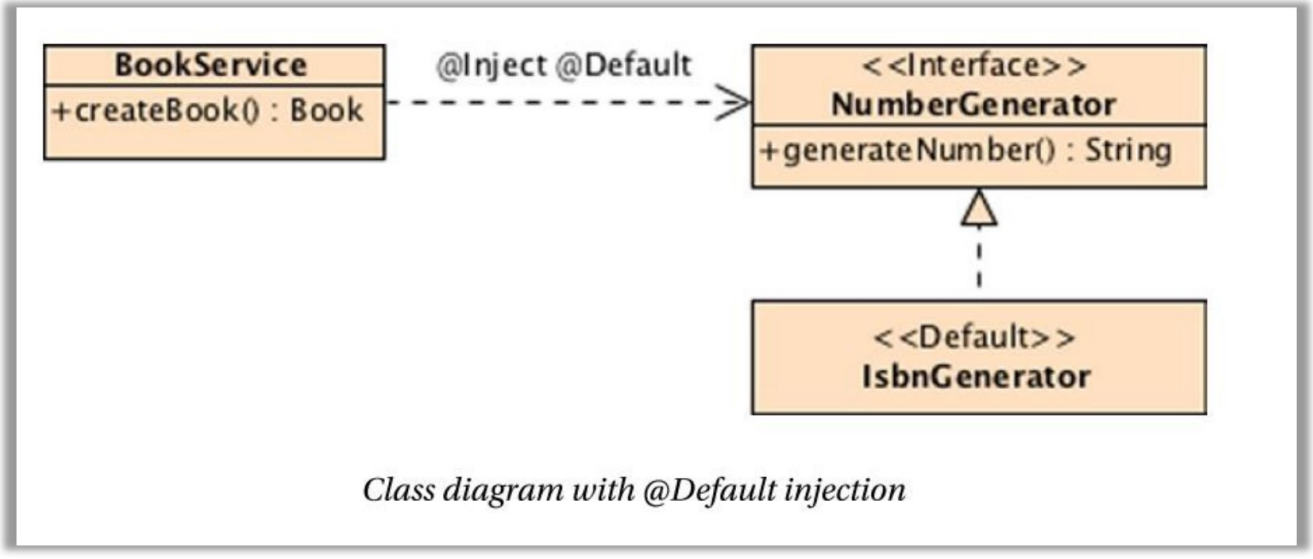
```
@Inject
public BookService (NumberGenerator numberGenerator) {
  this.numberGenerator = numberGenerator;
}
```

Setter injection point

```
@Inject
public void setNumberGenerator(NumberGenerator numberGenerator) {
  this.numberGenerator = numberGenerator;
}
```

Default Injection

- عندما يكون هناك implementation وحيد لاية interface وعند القيام بعملية inject سيتم استعماله بشكل افتراضي بدون الحاجة لاستخدام اية qualifier .
- Container ستستعمل qualifier @Default للقيام بذلك دون ان نحتاج كتابته بشكل واضح.
- عند تعريف Bean دون تحديدQualifier سيتم تحديد qualifier @Default لها.



Class diagram with @Default injection

Default Injection

```
@Inject
private NumberGenerator numberGenerator;
```

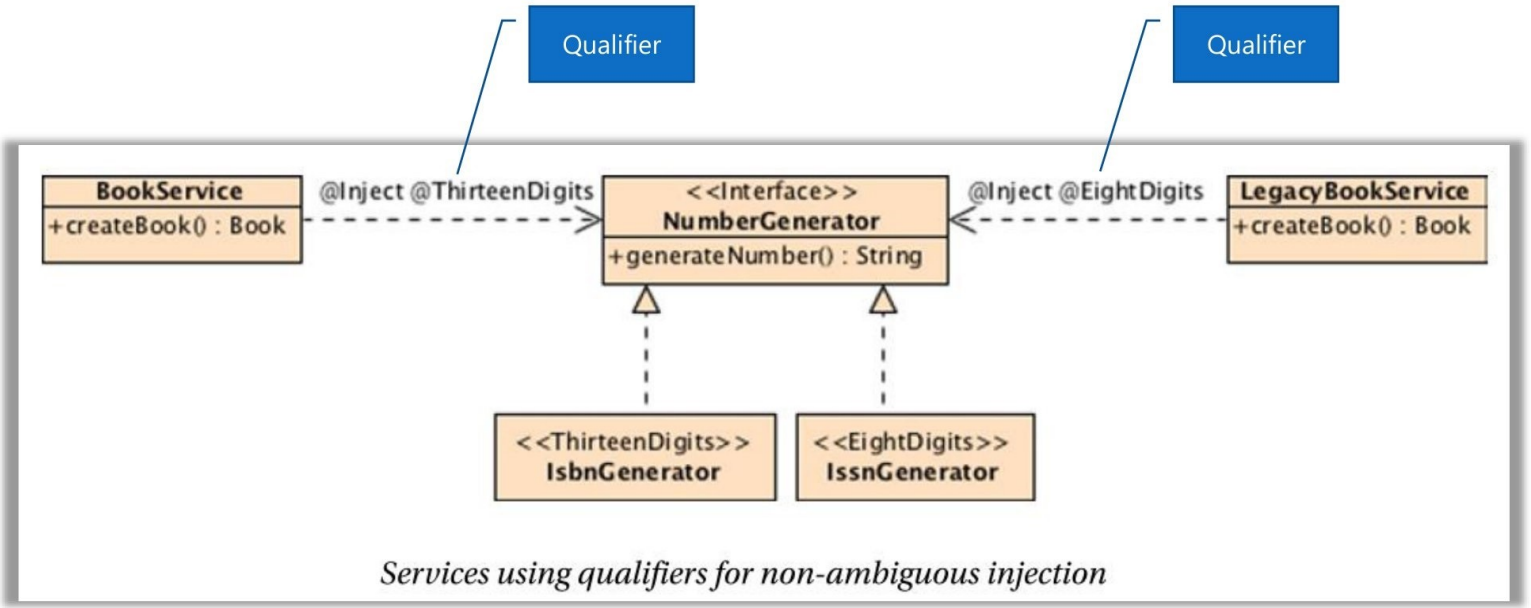
=

```
@Inject @Default
private NumberGenerator numberGenerator;
```

```
@Default
public class IsbnGenerator implements NumberGenerator {

  public String generateNumber() {
    return "13-84356-" + Math.abs(new Random().nextInt());
  }
}
```

Qualifiers

- عند استخدام interface في اكثر من implementation ونريد ان نقوم بعملية inject باستخدامه يتم استخدام qualifier للتمييز بين هذه implementations .



Services using qualifiers for non-ambiguous injection

Qualifiers

- يتم كتابة Qualifier باستخدام مجموعة من annotations ومن تم تستخدم في المكان الخاص بها.

```
@Qualifier
@Retention(RUNTIME)
@Target({FIELD, TYPE, METHOD})
public @interface ThirteenDigits { }
```

```
@Qualifier
@Retention(RUNTIME)
@Target({FIELD, TYPE, METHOD})
public @interface EightDigits { }
```

```
@ThirteenDigits
public class IsbnGenerator implements NumberGenerator {

  public String generateNumber() {
    return "13-84356-" + Math.abs(new Random().nextInt());
  }
}
```

```
@EightDigits
public class IssnGenerator implements NumberGenerator {

  public String generateNumber() {
    return "8-" + Math.abs(new Random().nextInt());
  }
}
```

Qualifiers

- تستخدم هذه Qualifiers لتحديد implementation الذي نريد من خلال استخدام annotation التي تم كتابتها.

```
public class BookService {

  @Inject @ThirteenDigits
  private NumberGenerator numberGenerator;

  public Book createBook(String title, Float price, String description) {
    Book book = new Book(title, price, description);
    book.setIsbn(numberGenerator.generateNumber());
    return book;
  }
}
```

```
public class LegacyBookService {

  @Inject @EightDigits
  private NumberGenerator numberGenerator;

  public Book createBook(String title, Float price, String description) {
    Book book = new Book(title, price, description);
    book.setIsbn(numberGenerator.generateNumber());
    return book;
  }
}
```

beans.xml

- هو bean archive descriptor التي عن طريقه يتم تحديد discovery mode الخاص ب beans ويحتوي على كل من Children التالية:
- Interceptors
- Decorators
- Alternatives

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://xmlns.jcp.org/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
       http://xmlns.jcp.org/xml/ns/javaee/beans_2_0.xsd"
       bean-discovery-mode="all">

</beans>
```

Discovery Mode Example

- المثال التالي يبين استخدام discovery mode .

```java
public class UserBean {

    public UserBean() {
    }

    private String userName = "Ali";

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }

}
```

```java
@Named
public class DiscoverBean {

    @Inject
    UserBean userBean;

    private String userName;

    public String getUserName() {
        return userBean.getUserName();
    }

}
```

Discovery Mode Example

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Producer Example</title>
    </h:head>
    <h:body>

        <h1>User Name is: #{discoverBean.userName}</h1>

    </h:body>
</html>
```

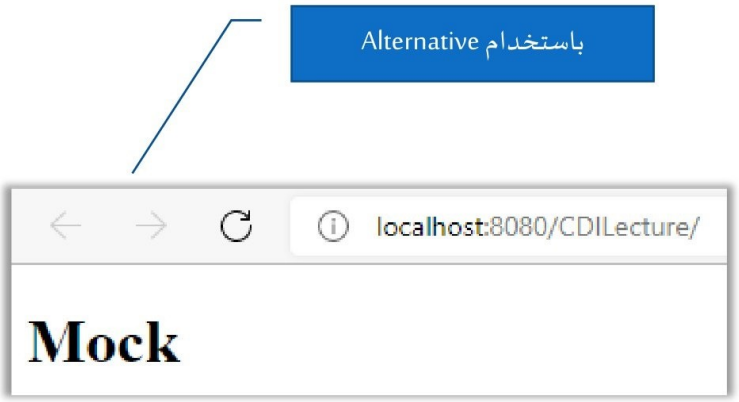
Alternatives

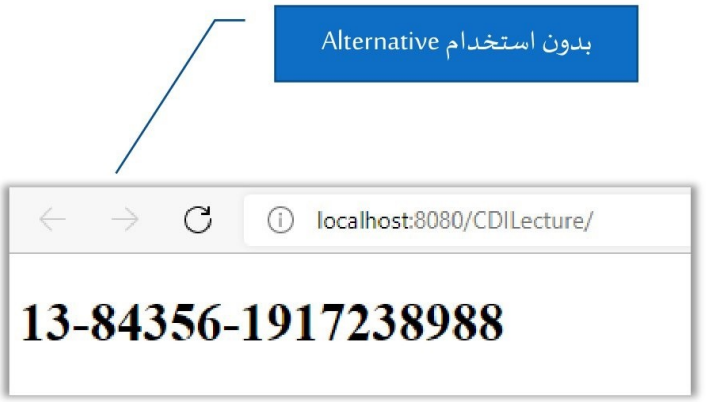
- تسمح Qualifiers بالاختيار بين مجموعة من implementations الخاصة ب interface ، لكن احياناً نحتاج ان نختار implementation بالاعتماد على deployment scenario يمكن القيام بذلك عن طريق استخدام @Alternative annotation ، مع عمل activation لها في ملف bean.xml .

```
@Alternative
public class MockGenerator implements NumberGenerator {

  @Override
  public String generateNumber() {
    return "Mock";
  }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://xmlns.jcp.org/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
       http://xmlns.jcp.org/xml/ns/javaee/beans_2_0.xsd"
       bean-discovery-mode="all">

    <alternatives>
        <class>ly.alaman.cdilecture.MockGenerator</class>
    </alternatives>

</beans>
```

Alternatives





@Producers

- عند الحاجة لعمل inject لاي primitive type او array types او not CDI Enabled POJO يتم استخدام producer .

```
public class NumberProducer {

  @Produces @ThirteenDigits
  private String prefix13digits = "13-";

  @Produces @ThirteenDigits
  private int editorNumber = 84356;

  @Produces @Random
  public double random() {
    return Math.abs(new Random().nextInt());
  }
}
```

```
@ThirteenDigits
public class IsbnGenerator implements NumberGenerator {

  @Inject @ThirteenDigits
  private String prefix;

  @Inject @ThirteenDigits
  private int editorNumber;

  @Inject @Random
  private double postfix;

  public String generateNumber() {
    return prefix + editorNumber + postfix;
  }
}
```

Interceptors

- تسمح لل container بتنفيذ بعض business logic قبل ان يتم استدعاء دالة من Bean .
- يمكن استخدام اكثر من interceptor .

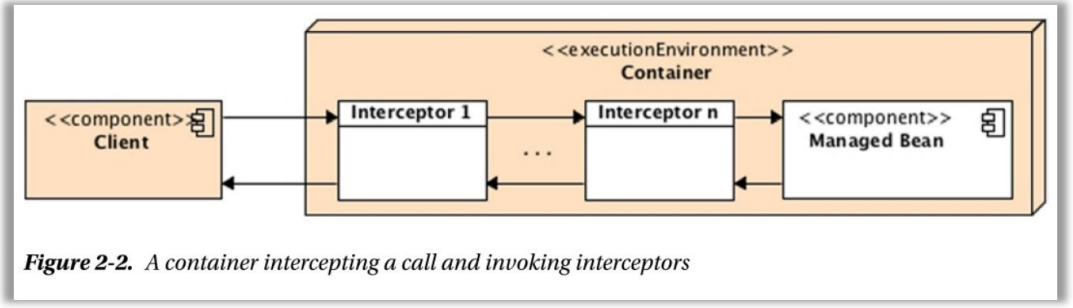


Figure 2-2. *A container intercepting a call and invoking interceptors*

Class Interceptors

- البرنامج التالي يبين استخدام Interceptor .

```
@Interceptor
public class LoggingInterceptor {

    public LoggingInterceptor() {
    }

    @AroundInvoke
    public Object logMethod(InvocationContext ic) throws Exception {

        System.out.println("Method intercepted: " + ic.getMethod().getName());
        return ic.proceed();
    }

}
```

Activating CDI interceptors

- يتم عمل activation ل interceptor من خلال bean.xml .

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://xmlns.jcp.org/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/beans_2_0.xsd"
       bean-discovery-mode="all">

    <interceptors>
        <class>ly.alaman.cdilecture.LoggingInterceptor</class>
    </interceptors>
</beans>
```

@Using Interceptors

- يتم استخدام interceptor من خلال @Interceptors annotation مع تحديد implementation class الخاصة به.

```
@Named(value = "bookBean")
@RequestScoped
@Interceptors(LoggingInterceptor.class)
public class BookBean {

    public BookBean() {
    }

    public String showBook() {
        return "book_details";
    }

}
```

Interceptor Binding

- تستخدم لاضافة interceptor دون الحاجة لتحديد interceptor class من خلال استخدام @InterceptorBinding annotation.

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

```
@Interceptor
@Loggable
public class LoggingInterceptor {

    public LoggingInterceptor() {
    }

    @AroundInvoke
    public Object logMethod(InvocationContext ic) throws Exception {

        System.out.println("Method Intercepted: " + ic.getMethod().getName());
        return ic.proceed();
    }

}
```

Interceptor Binding

Java EE

باستخدام Interceptor Binding

```
@Named(value = "bookBean")
@RequestScoped
@Loggable
public class BookBean {

    public BookBean() {
    }

    @Inject
    private NumberGenerator numberGenerator;

    public NumberGenerator getNumberGenerator() {
        return numberGenerator;
    }

    public String showBook() {
        return "book_details";
    }
```

بدون باستخدام Interceptor Binding

```
@Named(value = "bookBean")
@RequestScoped
@Interceptors(LoggingInterceptor.class)
public class BookBean {

    public BookBean() {
    }

    @Inject
    private NumberGenerator numberGenerator;

    public NumberGenerator getNumberGenerator() {
        return numberGenerator;
    }

    public String showBook() {
        return "book_details";
    }

}
```

Decorators

- تستخدم في إضافة بعض logic إلى business method من خلال استخدام Decorator pattern .
- يتم الحصول على decorator class باستخدام @Decoartor and @Delgate annotations .
- تم تفعيلها في bean.xml .

Decorators

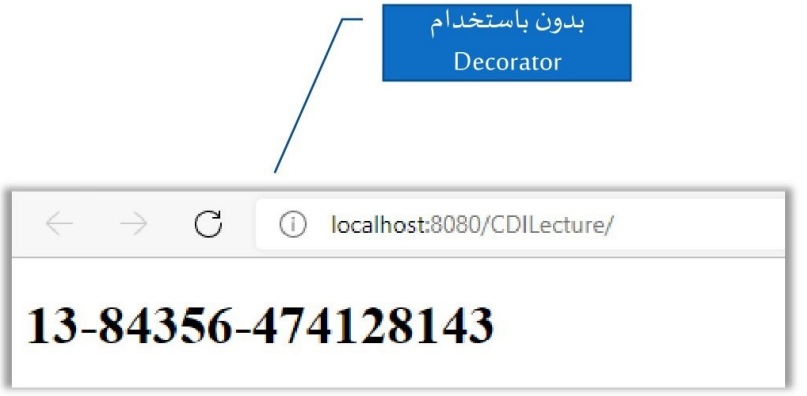
- المثال التالي يبين كيفية الإضافة على IsbnGenerator class من خلال استخدام IsbnGeneratorDecorator class .

```java
@Decorator
public class IsbnGeneratorDecorator implements NumberGenerator {

  @Inject @Delegate
  private NumberGenerator numberGenerator;

  @Override
  public String generateNumber() {
    return numberGenerator.generateNumber()+"-ISBN";
  }
}
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://xmlns.jcp.org/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                           http://xmlns.jcp.org/xml/ns/javaee/beans_2_0.xsd"
       bean-discovery-mode="all">

    <decorators>
        <class>ly.alaman.cdilecture.IsbnGeneratorDecorator</class>
    </decorators>

</beans>
```

Decorators


باستخدام Decorator
localhost:8080/CDILecture/
13-84356-1411289503-ISBN


بدون باستخدام
Decorator
localhost:8080/CDILecture/
13-84356-474128143