EJB 3.0 provides specification to intercept business methods calls using methods annotated with @AroundInvoke annotation. An interceptor method is called by ejbContainer before business method call it is intercepting. Following is the example signature of an interceptor method

```
@AroundInvoke
public Object methodInterceptor(InvocationContext ctx) throws Exception
{
    System.out.println("*** Intercepting call to LibraryBean method: "
    + ctx.getMethod().getName());
    return ctx.proceed();
}
```

Interceptor methods can be applied or bound at three levels

- **Default** - Default interceptor is invoked for every bean within deployment. Default interceptor can be applied only via xml ejb-jar.xml.
- **Class** - Class level interceptor is invoked for every method of the bean. Class level interceptor can be applied both by annotation of via xml ejb-jar.xml.
- **Method** - Method level interceptor is invoked for a particular method of the bean. Method level interceptor can be applied both by annotation of via xml ejb-jar.xml.

We are discussing Class level interceptor here.

**Interceptor class**

```
package com.tutorialspoint.interceptor;

import javax.interceptor.AroundInvoke;
import javax.interceptor.InvocationContext;

public class BusinessInterceptor {
    @AroundInvoke
    public Object methodInterceptor(InvocationContext ctx) throws Exception
    {
        System.out.println("*** Intercepting call to LibraryBean method: "
        + ctx.getMethod().getName());
        return ctx.proceed();
    }
}
```

**Remote Interface**

```
import javax.ejb.Remote;

@Remote
public interface LibraryBeanRemote {
    //add business method declarations
}
```

**Intercepted Stateless EJB**

```
@Interceptors (new {BusinessInterceptor.class})
@Stateless
public class LibraryBean implements LibraryBeanRemote {
    //implement business method
```
**Example Application**

Let us create a test EJB application to test intercepted stateless EJB.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <code>EjbComponent</code> under a package <code>com.tutorialspoint.interceptor</code> as explained in the EJB - Create Application chapter. You can also use the project created in EJB - Create Application chapter as such for this chapter to understand intercepted ejb concepts.</td>
</tr>
<tr>
<td>2</td>
<td>Create <code>LibraryBean.java</code> and <code>LibraryBeanRemote</code> under package <code>com.tutorialspoint.interceptor</code> as explained in the EJB - Create Application chapter. Keep rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Clean and Build the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>4</td>
<td>Finally, deploy the application in the form of jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet.</td>
</tr>
<tr>
<td>5</td>
<td>Now create the ejb client, a console based application in the same way as explained in the EJB - Create Application chapter under topic Create Client to access EJB.</td>
</tr>
</tbody>
</table>

**EJBComponent EJBModule**

**LibraryBeanRemote.java**

```java
package com.tutorialspoint.interceptor;

import java.util.List;
import javax.ejb.Remote;

@Remote
public interface LibraryBeanRemote {
    void addBook(String bookName);
    List getBooks();
}
```

**LibraryBean.java**

```java
package com.tutorialspoint.interceptor;

import java.util.ArrayList;
import java.util.List;
import javax.ejb.Stateless;
import javax.interceptor.Interceptors;

@Interceptors({BusinessInterceptor.class})
@Stateless
public class LibraryBean implements LibraryBeanRemote {
    List<String> bookShelf;

    public LibraryBean(){
        bookShelf = new ArrayList<String>();
    }

    public void addBook(String bookName) {
        bookShelf.add(bookName);
    }
}
public List<String> getBooks() {
    return bookShelf;
}

- As soon as you deploy the EjbComponent project on JBOSS, notice the jboss log.
- JBoss has automatically created a JNDI entry for our session bean - LibraryBean/remote.
- We'll using this lookup string to get remote business object of type - com.tutorialspoint.interceptor.LibraryBeanRemote

**JBoss Application server log output**

```
16:30:01,401 INFO  [JndiSessionRegistrarBase] Binding the following Entries in Global JNDI:
   LibraryBean/remote - EJB3.x Default Remote Business Interface
   LibraryBean/remote-com.tutorialspoint.interceptor.LibraryBeanRemote - EJB3.x Remote Business Interface

16:30:02,723 INFO  [JndiSessionRegistrarBase] Binding the following Entries in Global JNDI:
   LibraryBean/remote - EJB3.x Default Remote Business Interface
   LibraryBean/remote-com.tutorialspoint.interceptor.LibraryBeanRemote - EJB3.x Remote Business Interface
```

**EJBTester EJBClient**

**jndi.properties**

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
java.naming.provider.url=localhost
```

- These properties are used to initialize the InitialContext object of java naming service
- InitialContext object will be used to lookup stateless session bean

**EJBTester.java**

```java
package com.tutorialspoint.test;

import com.tutorialspoint.stateful.LibraryBeanRemote;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Properties;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {

    BufferedReader brConsoleReader = null;
    Properties props;
    InitialContext ctx;
    {
        props = new Properties();
        try {
```
```java
class EJBTester {
    private void showGUI()
    {
        System.out.println("*************************");
        System.out.println("Welcome to Book Store");
        System.out.println("*************************");
        System.out.print("Options 
1. Add Book
2. Exit 
Enter Choice: ");
    }

    private void testInterceptedEjb()
    {
        try {
            int choice = 1;
            LibraryBeanRemote libraryBean =
            ctx.lookup("LibraryBean/remote");

            while (choice != 2) {
                String bookName; showGUI();
                String strChoice = brConsoleReader.readLine();
                choice = Integer.parseInt(strChoice);
                if (choice == 1) {
                    System.out.print("Enter book name: ");
                    bookName = brConsoleReader.readLine();
                    Book book = new Book();
                    book.setName(bookName);
                    libraryBean.addBook(book);
                } else if (choice == 2) {
                    break;
                }
            }

            List<Book> booksList = libraryBean.getBooks();

            System.out.println("Book(s) entered so far: "+ booksList.size());
            int i = 0;
            for (Book book:booksList) {
                System.out.println((i+1)+". "+ book.getName());
                i++;
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        } finally {
            try {
                if(brConsoleReader !=null){
                    brConsoleReader.close();
                }
            } catch (IOException ex) {
                System.out.println(ex.getMessage());
            }
        }
    }

    public static void main(String[] args) {
        EJBTester ejbTester = new EJBTester();
        ejbTester.testInterceptedEjb();
    }
}
```
EJBTester is doing the following tasks.

- Load properties from jndi.properties and initialize the InitialContext object.
- In testInterceptedEjb method, jndi lookup is done with name - "LibraryBean/remote" to obtain the remote business object statelessejb.
- Then user is shown a library store User Interface and he/she is asked to enter choice.
- If user enters 1, system asks for book name and saves the book using stateless session bean addBook method. Session Bean is storing the book in its instance variable.
- If user enters 2, system retrieves books using stateless session bean getBooks method and exits.

**Run Client to access EJB**

Locate EJBTester.java in project explorer. Right click on EJBTester class and select **run file**.

Verify the following output in Netbeans console.

```
run:
Welcome to Book Store
Options
1. Add Book
2. Exit
Enter Choice: 1
Enter book name: Learn Java
Welcome to Book Store
Options
1. Add Book
2. Exit
Enter Choice: 2
Book(s) entered so far: 1
1. Learn Java
BUILD SUCCESSFUL (total time: 13 seconds)
```

**JBoss Application server log output**

Verify the following output in JBoss Application server log output.

```
```

Loading [MathJax]/jax/output/HTML-CSS/jax.js