- The argument and return value types for a method must be legal types for JAX-WS if the method is a web service method or corresponds to a method on the session bean's web service endpoint.

- The `throws` clause may define arbitrary application exceptions.

Note: Callback methods are permitted to have `public` access type. This raises the question of whether a callback method can also be exposed as a business method through one or more client views. Doing so is not prohibited, but should be done with caution. The runtime context (e.g. transaction context, caller principal, operations allowed, etc.) for a method invoked as a callback can differ significantly from the context for the same method when invoked via a client invocation. As a general rule, callback methods should not be exposed as business methods. Therefore, it is recommended that all non-business methods be assigned an access type other than `public`.

Compatibility Note: EJB 1.0 allowed the business methods to throw the `java.rmi.RemoteException` to indicate a non-application exception. This practice was deprecated in EJB 1.1—an EJB 1.1 or EJB 2.0 or later compliant enterprise bean should throw the `javax.ejb.EJBException` or another `RuntimeException` to indicate non-application exceptions to the container (see Section 9.2.2). An EJB 2.0 or later compliant enterprise bean should not throw the `java.rmi.RemoteException` from a business method.

4.9.7 Session Bean's Business Interface

The following are the requirements for the session bean's business interface:

- The interface must not extend the `javax.ejb.EJBObject` or `javax.ejb.EJBLocalObject` interface.

- If the business interface is a remote business interface, the argument and return values must be of valid types for RMI/IIOP. The remote business interface is not required or expected to be a `java.rmi.Remote` interface. The `throws` clause should not include the `java.rmi.RemoteException`. The methods of the business interface may only throw the `java.rmi.RemoteException` if the interface extends `java.rmi.Remote`.

- The interface is allowed to have superinterfaces.

- If the interface is a remote business interface, its methods must not expose local interface types, timers or timer handles as arguments or results.

- The bean class must implement the interface or the interface must be designated as a local or remote business interface of the bean by means of the `Local` or `Remote` annotation or in the deployment descriptor. The following rules apply:
    - A bean class is permitted to have more than one interface.
    - If the bean does not expose a no-interface view and has one or more interface that is not in the exclude list of interfaces listed below and is not designated explicitly as a local or a remote interface by use of the `Local` or the `Remote` annotation on the

bean class or the interface, or by means of the deployment descriptor, all these interfaces are assumed to be local business interfaces of the bean.

- If the bean class has one or more interface that is not in the exclude list of interfaces listed below and is not designated explicitly as a local or a remote interface by use of the `Local` or the `Remote` annotation on the bean class or the interface, or by means of the deployment descriptor, and the bean class is annotated with the `Local` annotation, all its interfaces are assumed to be local business interfaces of the bean.
- If the bean class has one or more interface that is not in the exclude list of interfaces listed below and is not designated explicitly as a local or a remote interface by use of the `Local` or the `Remote` annotation on the bean class or the interface, or by means of the deployment descriptor, and the bean class is annotated with the `Remote` annotation, all its interfaces are assumed to be remote business interfaces of the bean.
- If a bean class has more than one interface—excluding the interfaces listed below—and any interface of the bean class is explicitly designated as a business interface of the bean by means of the `Local` or `Remote` annotation on the bean class or interface or in the deployment descriptor, or the bean exposes a no-interface view, all business interfaces must be explicitly designated as such.
- The following interfaces are excluded when determining whether the bean class has more than one interface: `java.io.Serializable`; `java.io.Externalizable`; any of the interfaces defined by the `javax.ejb` package.
- The same business interface cannot be both a local and a remote business interface of the bean.[29]
- While it is expected that the bean class will typically implement its business interface(s), if the bean class uses annotations or the deployment descriptor to designate its business interface(s), it is not required that the bean class also be specified as implementing the interface(s).

[29] It is also an error if the `Local` and/or `Remote` annotations are specified both on the bean class and on the referenced interface and the values differ.

The following examples assume that there is no deployment descriptor associated with the bean and neither the `Local` nor the `Remote` annotation is specified on the bean class or an interface unless noted.

Example 1 - session bean `A` exposes *two local* business interfaces, `Foo` and `Bar`:

```
public interface Foo { ... }

public interface Bar { ... }

@Stateless
public class A implements Foo, Bar { ... }
```

Example 2 - session bean `A` exposes *two local* business interfaces, `Foo` and `Bar`:

```
public interface Foo { ... }

public interface Bar { ... }

@Local
@Stateless
public class A implements Foo, Bar { ... }
```

Example 3 - session bean `A` exposes *two remote* business interfaces, `Foo` and `Bar`

```
public interface Foo { ... }

public interface Bar { ... }

@Remote
@Stateless
public class A implements Foo, Bar { ... }
```

Example 4 - session bean `A` exposes *ony one remote* business interface `Foo`

```
@Remote
public interface Foo { ... }

public interface Bar { ... }

@Stateless
public class A implements Foo, Bar { ... }
```

Example 5 - session bean `A` exposes *ony one remote* business interface `Foo`

```
public interface Foo { ... }

public interface Bar { ... }

@Remote(Foo.class)
@Stateless
public class A implements Foo, Bar { ... }
```

4.9.8 Session Bean's No-Interface View

The following are the requirements for a session bean that exposes a no-interface view: