Table of Contents

Every server-side action should be defined in a life cycle object (LCO). Sometimes, it's better to group functionality in helper objects.

It's very easy to encapsulate functionality in objects. Just create an object and add public methods:

Car.java

```
public class Car
{
    private String type;
    private int speed;

    public Car(String type, int speed)
    {
        this.type = type;
        this.speed = speed;
    }

    public String getType()
    {
        return sType;
    }

    public int getSpeed()
    {
         return speed;
    }
}
```

Our Car class has two methods: **getType** and **getSpeed**!

If you're using the Car class in your life cycle object, like the following:

```
// Application LCO
public class Application extends GenericBean
{
}

// Session LCO
public class Session extends Application
{
    public Car getCar()
    {
        Car car = (Car)get("car");

        if (car == null)
        {
            car = new Car();

            put("car", car);
```

```
        }

        return car;
    }
}
```

it will be possible to call getType and getSpeed.

```
connection.call("car", "getSpeed")
```

But this call would throw a SecurityException with “access is not allowed”.

It's not possible to call every method of an object just because it's public. This could open back doors.

You have to define accessible methods - from objects - via annotation:

Car.java

```
public class Car
{
    ...

    @Accessible
    public String getType()
    {
        return sType;
    }

    @Accessible
    public int getSpeed()
    {
         return speed;
    }
}
```

The call:

```
connection.call("car", "getSpeed");
```

will work without problems!

If you define a public method in your LCO (action), it's always accessible because, usually, you will offer business logic or storages for the client. But it's possible to deny the access to objects or actions:

Session.java

```
public class Session extends Application
{
    @NotAccessible
    public getCar()
```

```
    {
        ...
    }
}
```

The **NotAccessible** annotation denies the access via connection call but it's still possible to invoke the method directly in your Session LCO or a Screen LCO:

```
public class Session extends Application
{
    @NotAccessible
    public getCar()
    {
        ...
    }

    public int getSpeed()
    {
        return getCar().getSpeed();
    }
}

public class Screen extends Session
{
    public String getType()
    {
        return getCar().getType();
    }
}
```

The call:

```
connection.call("car", "getSpeed")
```

will fail with a SecurityException because car object is not accessible. The action call:

```
connection.callAction("getSpeed")
```

will work without problems.

The security controller doesn't check simple method calls because the developer should have the freedom to do everything on server side.