hybris software

An SAP Company

hybris software
An SAP Company

hybris Developer Training Part I - Core Platform

Programming with the hybris ServiceLayer

hybris software
An SAP Company

Architecture of the ServiceLayer

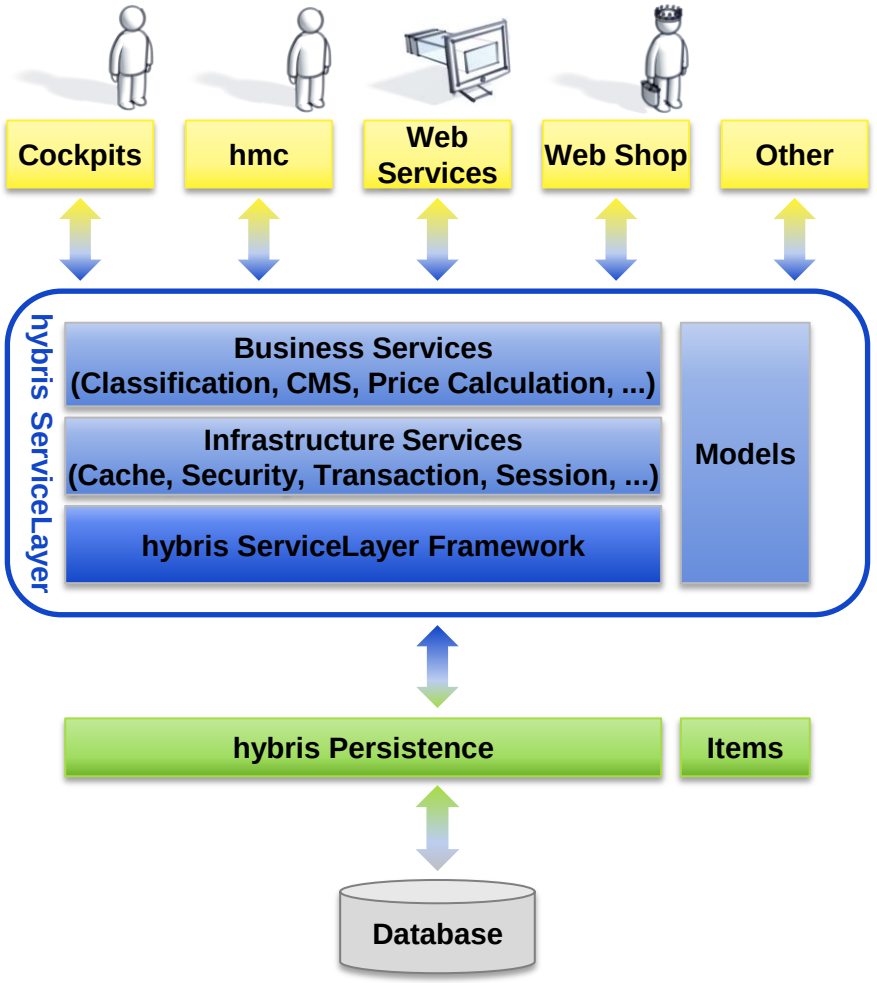
Models

Interceptors

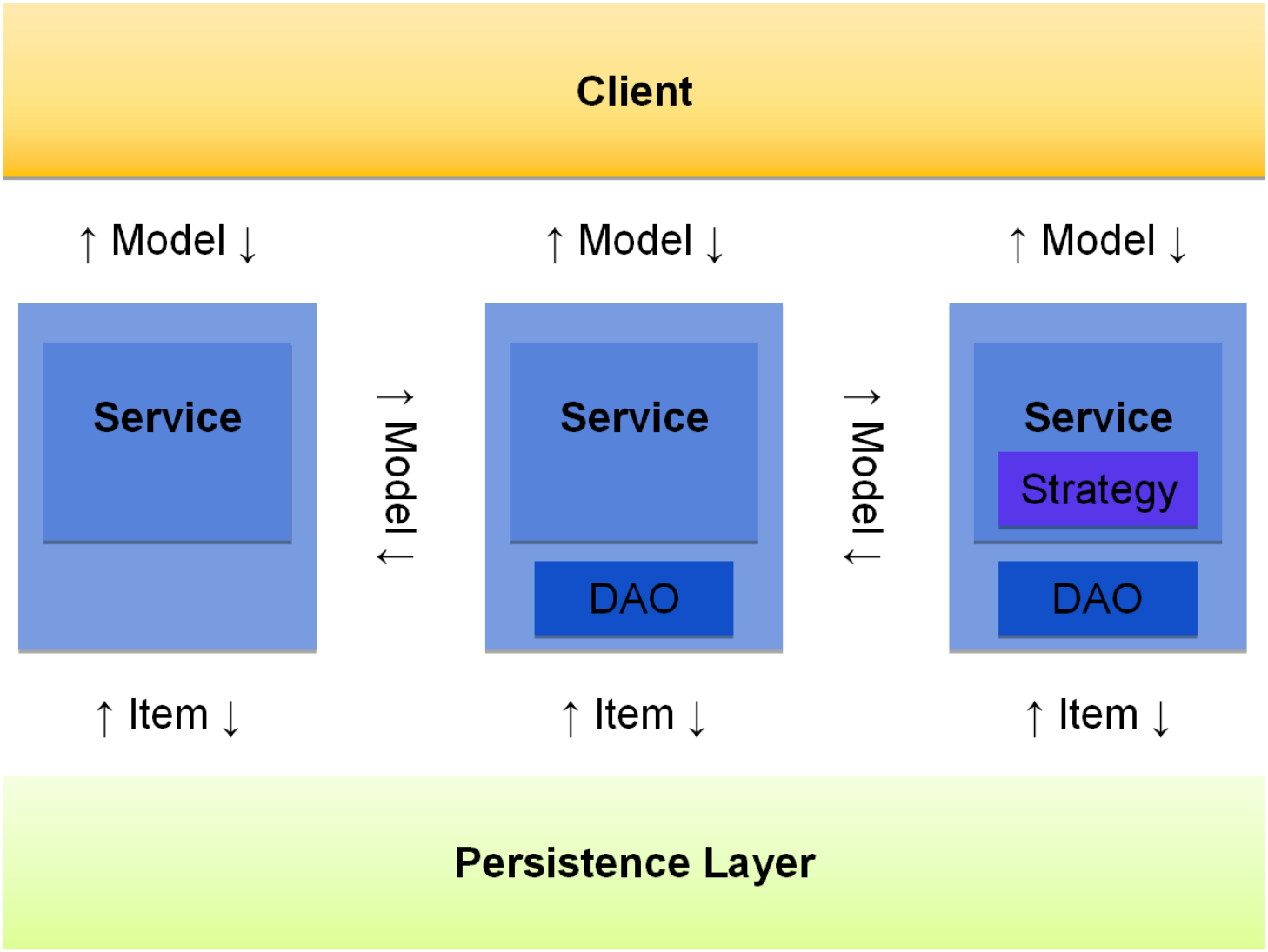
Beans generation

Overview of the hybris ServiceLayer

- The hybris architectural layer where you implement YOUR logic
- Provides a number of services, each with its well defined responsibilities
- Service oriented architecture based on Spring framework
- Provides hooks into modellife-cycle events for performing custom logic
- Provides a framework for publishing and receiving events



ServiceLayer – Structure and Data Objects



Using Services

- To implement your own business logic, you can:
  - use existing services
  - create your own services
  - replace existing services

- Each service is defined as a spring bean and has a spring alias
- To override an existing service re-alias it in the spring context

```
<alias alias="cartService"
       name="myCustomCartService" />
```

hybris software
An SAP Company

Architecture of the ServiceLayer

Models

Interceptors

Beans generation

Overview of Models (I)

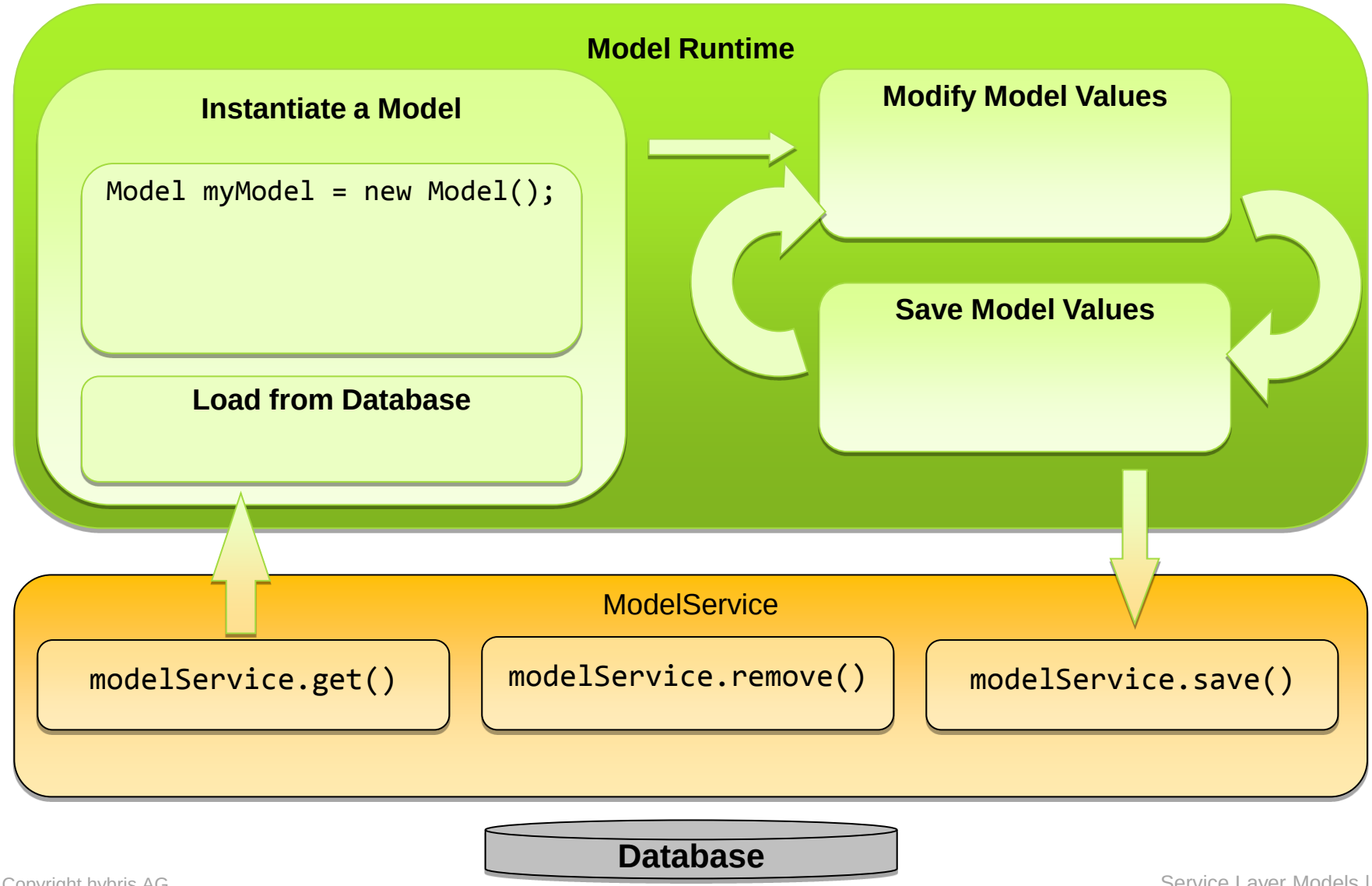
- Data objects the ServiceLayer is based on
- Each Item Type has a corresponding model class
- POJO-like objects
- Providing attributes with getter and setter methods
- Generated during build
  - `${HYBRIS_BIN_PATH}/platform/bootstrap/gensrc`

Overview of Models (II)

- Models represent a certain “snapshot” of data from the database
  - No attachment to database: representation is not live
  - When modifying a model, you must explicitly save it back
- You may influence loading of attributes
  - `servicelayer.prefetch` in advanced.properties

Never touch (hybris) models!

Lifecycle of a Model



Using Models

- The ModelService deals with all aspects of a model's life-cycle:
  - Loading models by PK
  - Creating models
  - Updating / saving models
  - Deleting models

- Factory Method:

```
ProductModel product = modelService.create(ProductModel.class);
```

- Constructor:

```
ProductModel product = new ProductModel();
```

hybris software
An SAP Company

Architecture of the ServiceLayer

Models

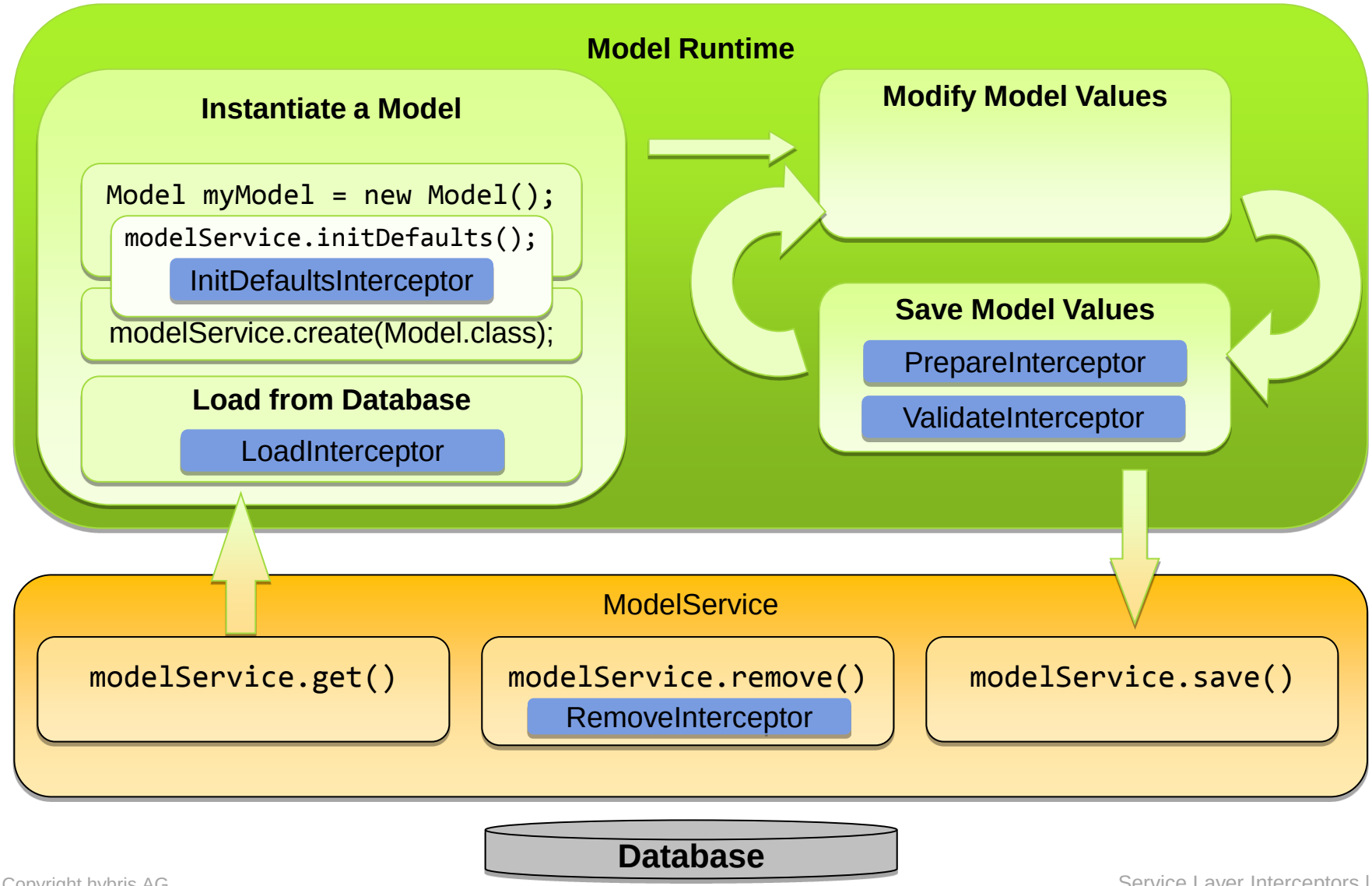
Interceptors

Beans generation

Interceptors Overview

- There are various types of interceptors allowing you to interrupt the intended course of a Model’s life cycle
- An interceptor addresses a particular step in a Model's life cycle
- With interceptors, you can modify a Model or raise exceptions to interrupt the current step e.g. certain values may be validated before a Model is saved
- An interceptor is registered as a Spring bean

Interceptors Overview - Lifecycle of a Model



Implementing interceptors

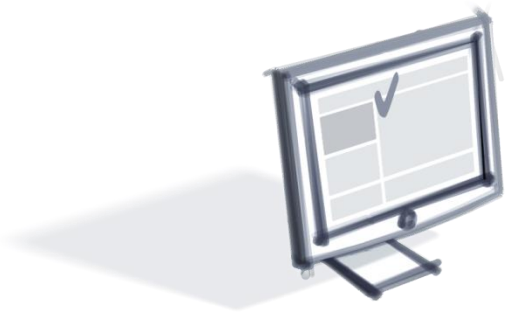
- To create an interceptor, one the following interfaces needs to be implemented:
  - `LoadInterceptor` is called whenever a model is loaded from the database
  - `InitDefaultsInterceptor` is called to fill a model with its default values
  - `PrepareInterceptor` is called before a model is saved to the database and before it is validated by `ValidateInterceptor`
  - `ValidateInterceptor` is called before a model is saved to the database and after it has been prepared by the `PrepareInterceptor`
  - `RemoveInterceptor` is called before a model is removed from the database

Implementation - Registering an interceptor

After implementing an interceptor, the interceptor is registered as a spring bean in myextension-spring.xml:

```
<bean id="myLoadInterceptor"
            class="my.package.MyLoadInterceptor"/>
```

```
public class MyLoadInterceptor implements LoadInterceptor
{
    public void onLoad(Object model, InterceptorContext ctx)
                throws InterceptorException
    {
          ....
    }
}
```



Implementation – mapping the interceptor

The interceptor is then mapped in *myextension*-spring.xml:

```
<bean id="myLoadMapping"
      class="de...interceptor.impl.InterceptorMapping">

   <property name="interceptor" ref="myLoadInterceptor"/>
   <property name="typeCode" value="Stadium"/>
   <property name="order" value="1"/>
   <property name="replacedInterceptors" ref="a,b,c"/>
</bean>
```

hybris software
An SAP Company

Architecture of the ServiceLayer

Models

Interceptors

Beans generation

Overview

- Java Beans are used as transport objects for the frontend layer
- Contain an abstraction of models (subset of attributes from models)
- Are automatically generated
- Their class name, attributes or superclass are defined in xml configuration file
- Each extension may provide a bean configuration file
- Definitions of these Java Beans or Enums are merged across extensions.

A declarative approach

- Generate Java Beans out of a configuration file

```
<bean class="org.training.data.MyPojo">
    <property name="id" type="String"/>
</bean>
```

```
public class MyPojo implements java.io.Serializable
{
    private String id;
    public MyPojo() {} //default constructor
    public String getId() {...}
    public void setId(String pId) {}
}
```

Why?

- Merge attributes into existing beans
- Useful as DataObjects used by Frontend layer

extension1-beans.xml

```
<bean class="org.training.data.MyPojo">
    <property name="id" type="String"/>
</bean>
```

extension2-beans.xml

```
<bean class="org.training.data.MyPojo">
    <property name="flag" type="boolean"/>
</bean>
```

Triggered through: `ant all`

```
public class MyPojo implements
java.io.Serializable
{
    private String id;
    private boolean flag;
    public MyPojo() {}
    public String getId() {...}
    public boolean getFlag() {...}
    public void setId(String pId) {...}
    public void setFlag(boolean pFlag) {...}
}
```

Generated to `platform/bootstrap/gensrc`

Templates

- Generation templates are velocity scripts

```
  <bean ... template=“resource/mypojo-
template.vm">
```

- Default templates:
  - bean: `global-beantemplate.vm`
  - enum: `global-enumtemplate.vm`
- Create a new template

my-template.vm

```
package $packageName;
#foreach ($i in $imports)
import $i;
#end
```

Templates implementation

Implementation

```
<bean class="org.training.data.MyPojoExtended"
      template="resources/my-template.vm”/>
        <property type="boolean" name="custom" />
        <property name="segment"
            type="org.training.enums.TestEnum"/>
</bean>
```

More info

wiki.hybris.com/display/release5/
Generating+Beans+and+Enums

Quiz-Questions

1. What is the hybris ServiceLayer?
2. Name 3 services
3. What is a ModelService and why should you use it?
4. How to create an interceptor?