SAP Customer Experience

Services

SAP Commerce Cloud Developer Training

THE BEST RUN

The Context

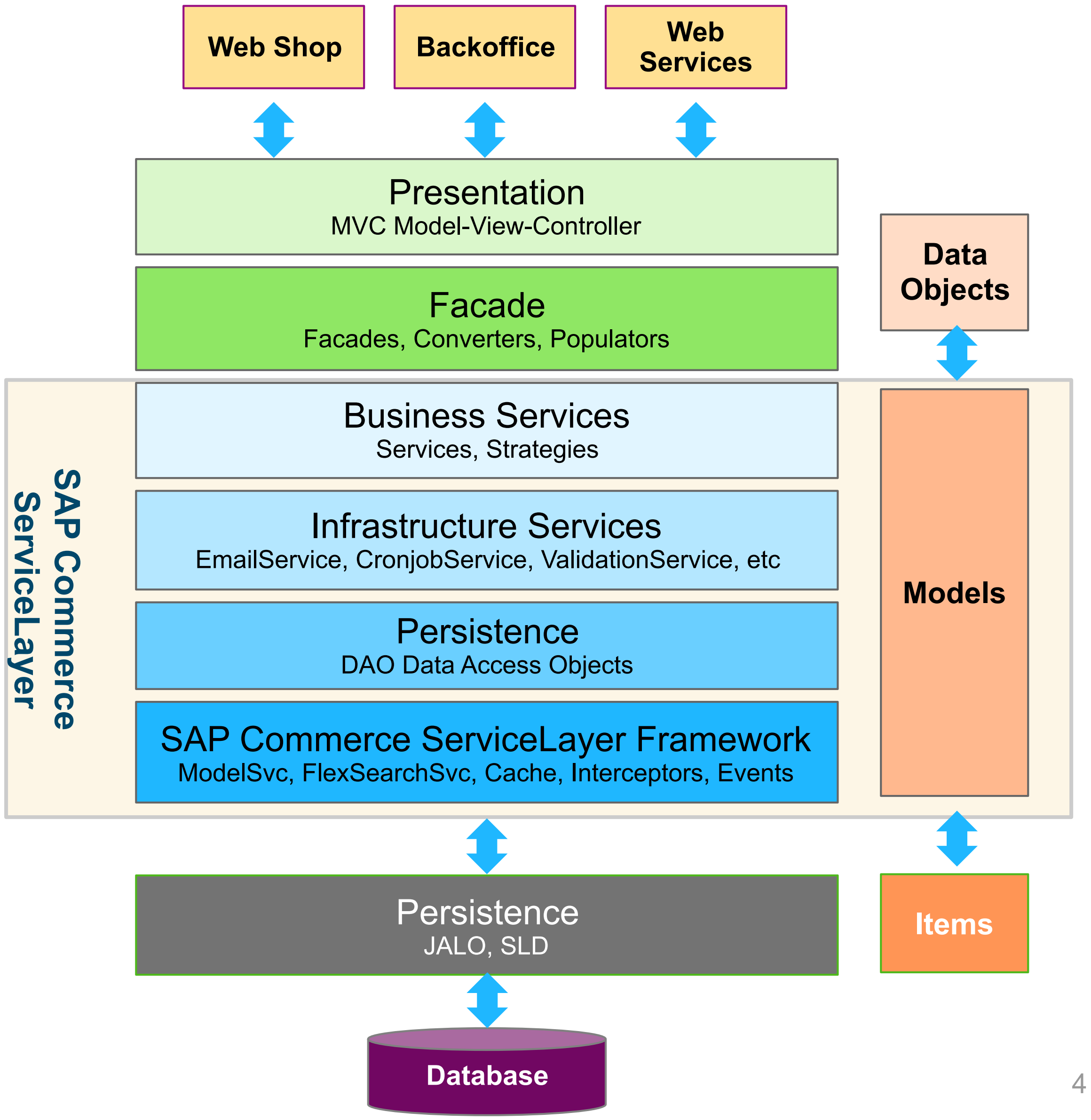
The SAP Commerce ServiceLayer is an **API** for developing services for SAP Commerce. It provides a number of common services, which you can extend, or develop your own.

Architecture of the ServiceLayer

ServiceLayer
Models
Interceptors

Overview of the SAP Commerce ServiceLayer

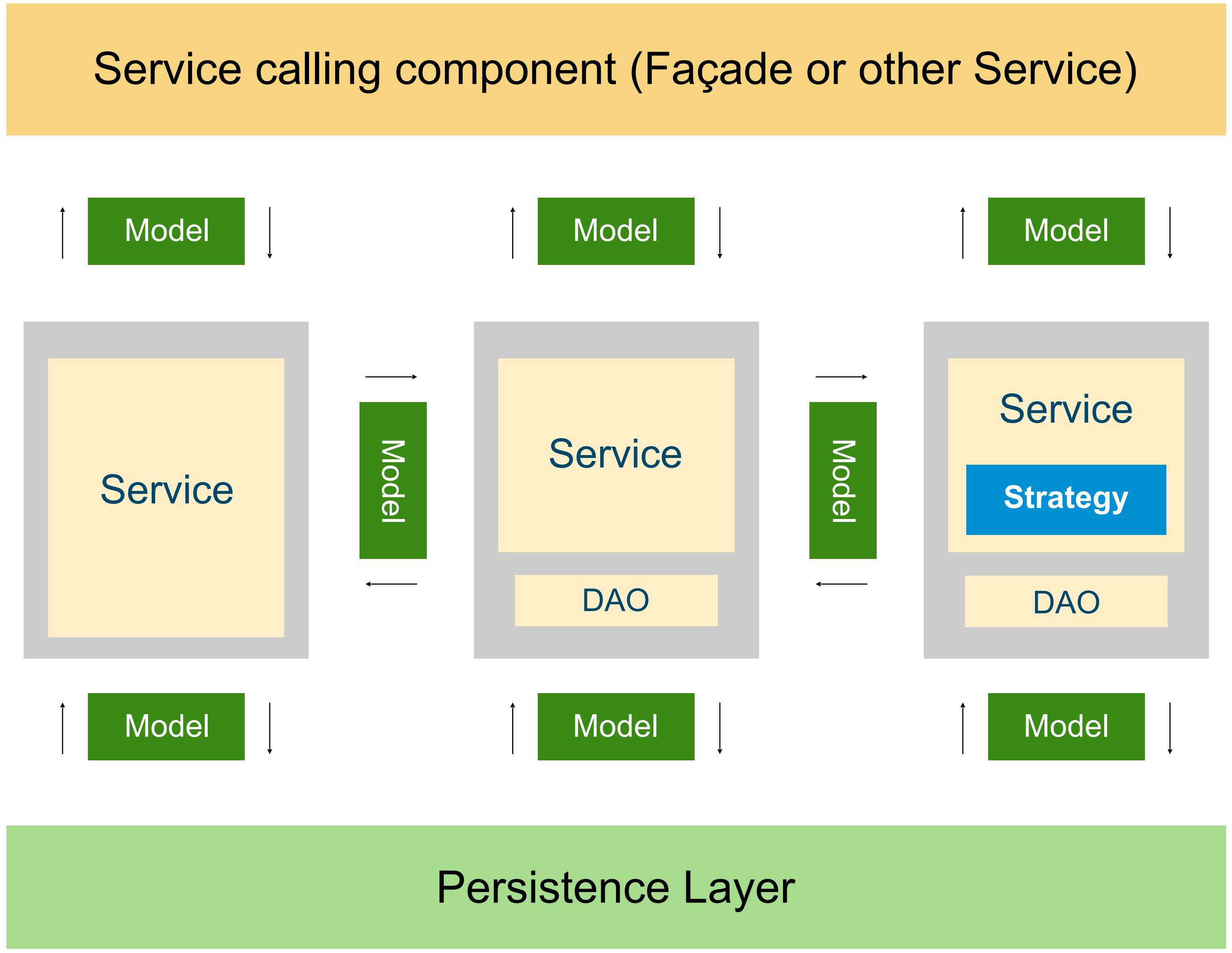
- The SAP Commerce architectural layer where you implement *your* logic
- Provides a number of services, each with well-defined responsibilities
- Service-oriented architecture based on the Spring framework
- Provides hooks into model life-cycle events for performing custom logic
- Provides a framework for publishing and receiving events



Web Shop
Backoffice
Web Services
Presentation
MVC Model-View-Controller
Facade
Facades, Converters, Populators
Data Objects
SAP Commerce ServiceLayer
Business Services
Services, Strategies
Infrastructure Services
EmailService, CronjobService, ValidationService, etc
Persistence
DAO Data Access Objects
SAP Commerce ServiceLayer Framework
ModelSvc, FlexSearchSvc, Cache, Interceptors, Events
Models
Persistence
JALO, SLD
Items
Database

ServiceLayer – Structure and Data Objects

Models are not exposed to the storefront



Using Services

- To implement your own business logic, you can:
  - Use existing services as is
  - Create your own services
  - Replace/extend/override existing services

- Each service in SAP Commerce 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" />

<bean id="myCustomCartService"
      class="my.project.MyCustomCartService" />
```

Configuring Services

- Services frequently need to call on other services or components
- Instead of fully configuring a new service, use the **parent** argument to inherit configuration from the service bean it is extending
- You may override any property inherited from the parent, or use it as is
- Special syntax for overriding or extending parent bean list values
- See Reusing configuration from other beans in the Spring Essentials for SAP Commerce
  - found in your handouts folder, under *Optional Reading.*

Models

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

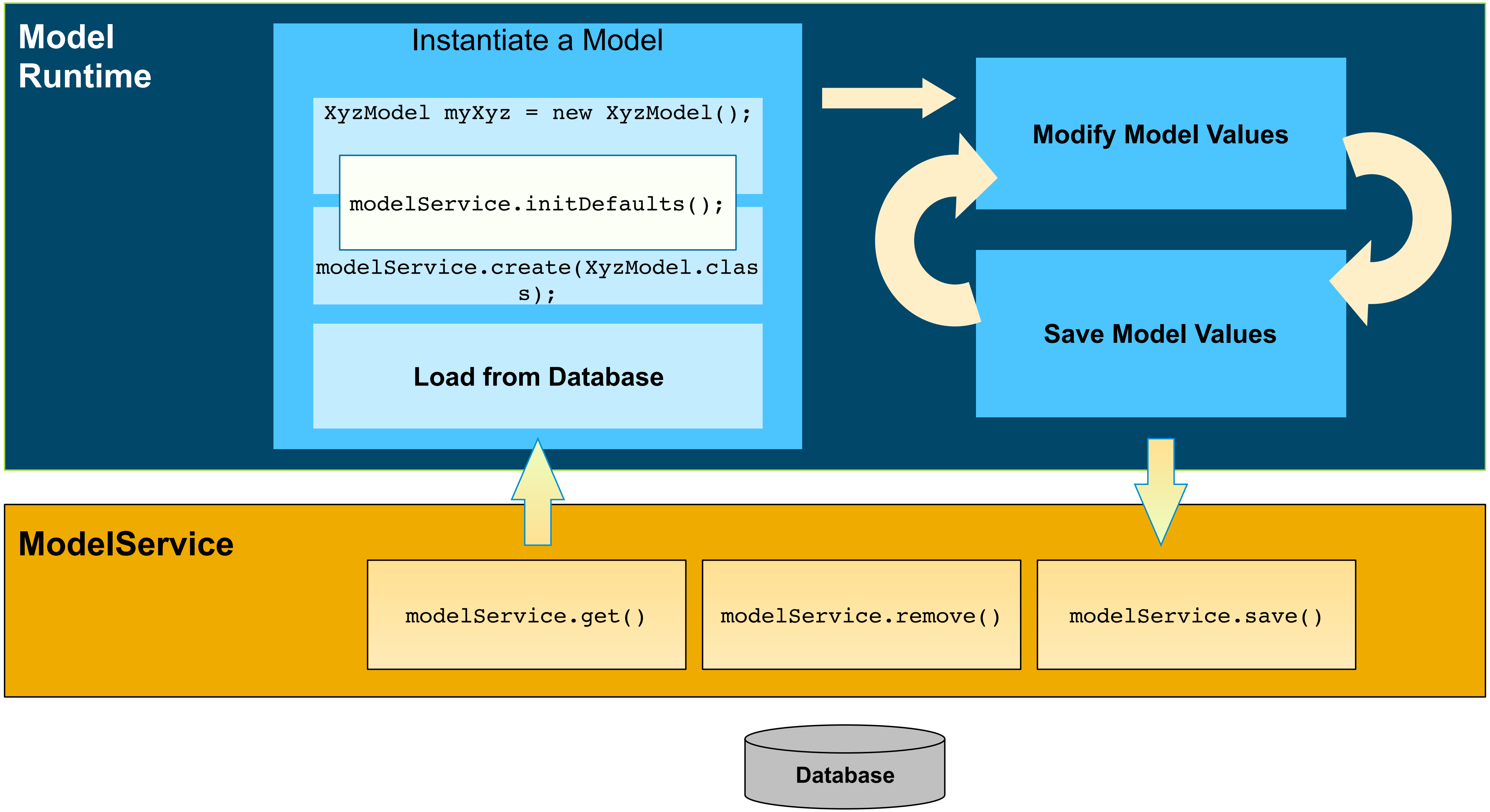
```
${HYBRIS_BIN_PATH}/platform/bootstrap/gensrc
```

Never manually edit SAP Commerce model classes!

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 `platform/resources/advanced.properties`

Lifecycle of a Model



Using Models

- The ModelService bean 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:

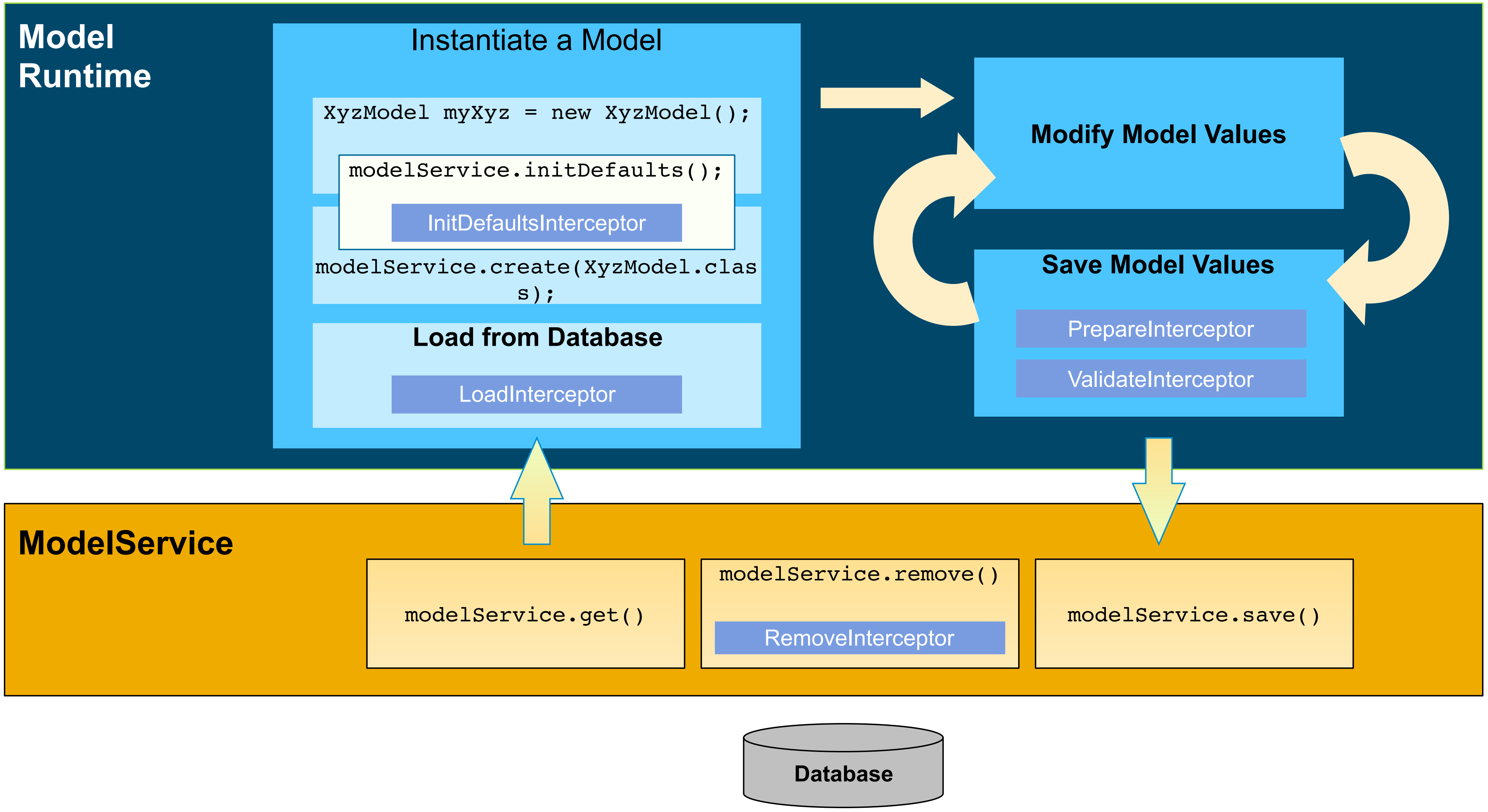
```
ProductModel product = new ProductModel();
```

Interceptors

Interceptors Overview

- Interceptors allow you to alter the course of a Model’s life cycle
- An interceptor targets one or more aspect of a Model's life cycle
- With interceptors, you can modify a Model or raise exceptions to interrupt the current step; for instance, a value may be validated before saving a Model
- An interceptor is registered as a Spring bean

Lifecycle of a Model



Implementing interceptors

- To create an interceptor, implement one or more of the following interfaces:
  - A `LoadInterceptor` is called whenever a model is loaded from the database
  - An `InitDefaultsInterceptor` is called to populate a model with default values
  - A `PrepareInterceptor` is called before a model is saved to the database but before it is validated by any `ValidateInterceptor`
  - A `ValidateInterceptor` is called before a model is saved to the database and after it has been prepared by a `PrepareInterceptor`
  - A `RemoveInterceptor` is called before an item 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.hybris.platform.servicelayer.interceptor.impl.InterceptorMapping">

   <property name="interceptor" ref="myLoadInterceptor"/>
   <property name="typeCode" value="Customer"/>
   <property name="order" value="1"/>
   <property name="replacedInterceptors"
             ref="existingInterceptorA,existingInterceptorB"/>
 </bean>
```

Disabling Interceptors

- This option can be used for relaxing certain constraints imposed on a data model in data integration scenarios, or for performance reasons.
- Interceptors can be disabled
  - Using API by calling the `sessionService.executeInLocalViewWithParams` method
  - Using ImpEx
- There are 3 attributes that allow you to disable specific interceptors
  - `disable.interceptor.beans`
  - `disable.interceptor.types`
  - `disable.UniqueAttributesValidator.for.types`

Key POINTS

The SAP Commerce architectural layer is where you implement ***your* logic**

Provides a service-oriented architecture based on the Spring framework

Use or replace existing services, or create your own

Models are POJO-like objects and generated during build in
`${HYBRIS_BIN_PATH}/platform/bootstrap/gensrc`

Use **`ModelService` for CRUD** operations on models

Use **Interceptors** to alter the course of a Model's lifecycle

Models are, by default, persisted without transactions

Keep junit tenant in sync with master tenant for reliable testing results

Testing is important, but for a 4-day class, please focus on getting hands-on with SAP Commerce

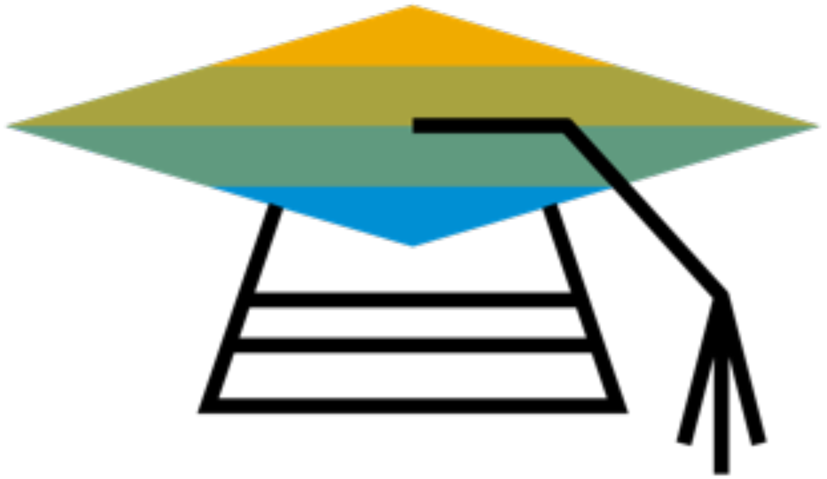
Don't waste too much time fixing your test environment if you're getting errors.

Exercise

Services

SAP Customer Experience

Thank you.



THE BEST RUN