A. Creating a RemoveInterceptor that Stores Deleted Users in a Separate Table.

1) Define an item that stores the data of each deleted user: items.xml

```
<itemtype code="UserAuditEntry" generate="true" autocreate="true">
                <deployment table="UserAuditEntries" typecode="8998"/>
                <attributes>
                        <attribute qualifier="uid" type="java.lang.String">
                                <persistence type="property"/>
                        </attribute>
                        <attribute qualifier="name" type="java.lang.String">
                                <persistence type="property"/>
                        </attribute>
                        <attribute qualifier="displayName" type="java.lang.String">
                                <persistence type="property"/>
                        </attribute>
                        <attribute qualifier="changeTimestamp" type="java.util.Date">
                                <persistence type="property"/>
                        </attribute>
                </attributes>
        </itemtype>
```

2) Create an interceptor that creates an instance of the above item each time a user is deleted:

```
public class AuditingUserRemoveInterceptor implements RemoveInterceptor
        {
                @Override
                public void onRemove(final Object o, final InterceptorContext ctx) throws
InterceptorException
                {
                        if (o instanceof UserModel)
                        {
                                final UserModel user = (UserModel) o;
                                final UserAuditEntryModel auditEntryModel =
ctx.getModelService().create(UserAuditEntryModel.class);
                                auditEntryModel.setChangeTimestamp(new Date());
                                auditEntryModel.setDisplayName(user.getDisplayName());
                                auditEntryModel.setName(user.getName());
                                auditEntryModel.setUid(user.getUid());
                                ctx.registerElementFor(auditEntryModel,
PersistenceOperation.SAVE);
                        }
                }
        }
```

3) Register the interceptor in the spring context using InterceptorMapping.

```
<bean id="MyValidateInterceptorMapping"
    class="de.hybris.platform.servicelayer.interceptor.impl.InterceptorMapping">
        <property name="interceptor" ref="myValidateInterceptor"/>
        <property name="typeCode" value="MyType"/>
        <property name="order" value="5000"/>
</bean>
```

B. Validating UserAuditEntryModels

Optionally, we can consider a more contrived scenario where we want to validate UserAuditEntryModels and only allow those where the username is not empty.

To achieve this goal, let's define the following ValidateInterceptor:

```
public class AuditEntryValidateInterceptor implements ValidateInterceptor
        {
                @Override
                public void onValidate(final Object o, final InterceptorContext ctx) throws
InterceptorException
                {
                        if (o instanceof UserAuditEntryModel)
                        {
                                final UserAuditEntryModel auditEntry = (UserAuditEntryModel) o;
                                if (StringUtils.isEmpty(auditEntry.getName()))
                                {
                                        throw new InterceptorException("User audit entries cannot
have empty username");
                                }
                        }
                }
        }
```

When this interceptor is registered, all <UserAuditEntries> created by the AuditingUserRemoveInterceptor are validated with the above interceptor. If the validation fails, all changes are rolled back (the user is not removed and <UserAuditEntry> is not created).