Requirement:

Identify how old the customer is in our site so that we can give some discount to those customers who are 3 years old.

How to achieve it?
We need to get the customer's registration date which we store as creation_time in DB and we need to get current time from Java Date API.

We need to do **current_time – creation_time** to know how old the customer is.

We can add a new attribute called "customer_site_age" as dynamic attribute.

Let's see the steps for the same.

Step 1:
Define a new attribute in the items.xml for Customer item type and make it as dynamic attribute.

Copy this code

```
<itemtype code="Customer" autocreate="false" generate="false">
            <description>Extending Customer type with additional attributes.</description>
    <attributes>
        <attribute autocreate="true" qualifier="customer_site_age" type="java.lang.Integer">
                    <modifiers read="true" write="true"/>
                    <persistence type="dynamic"/>
                <description>Dynamic attribute for customer site age</description>
        </attribute>
    </attributes>
</itemtype>
```

Here we have defined the persistent type as dynamic.

Step 2:
Define the Attributehandler which should be the bean id of the class which implements DynamicAttributeHandler.

Copy this code

```
<itemtype code="Customer" autocreate="false" generate="false">
            <description>Extending Customer type with additional attributes.</description>
    <attributes>
        <attribute autocreate="true" qualifier="customer_site_age" type="java.lang.Integer">
                <modifiers write="false"/>
                <persistence type="dynamic" attributeHandler="customerSiteAge"/>
                <description>Dynamic attribute for customer site age</description>
        </attribute>
    </attributes>
</itemtype>
```

We have added attributeHandler="CustomerSiteAge" in the persistent tag.

Step 3:
Define the class with custom logic

We can define the new class either by extending **AbstractDynamicAttributeHandler** or by implementing **DynamicAttributeHandler**

Define it in the trainingcore(your custom core) extension in org.training.core.attributes package

Copy this code

```
public class CustomerSiteAgeHandler extends
AbstractDynamicAttributeHandler<Integer, CustomerModel>
{
  @Override
    public Integer get(final CustomerModel model)
    {
        int customerSiteAge = 0;
        try
        {
            final Date customerRegisteredDate = model.getCreationtime();
            final Calendar cal = Calendar.getInstance();
            cal.setTime(customerRegisteredDate);
            final int registeredYear = cal.get(Calendar.YEAR);
            final int currentYear = Calendar.getInstance().get(Calendar.YEAR);
            customerSiteAge = currentYear - registeredYear;
        }
        catch (final Exception e)
        {
            e.printStackTrace();
        }
        return customerSiteAge;
    }
}
```

Step 4:
Associate the attribute handler specified in items.xml with spring bean id in trainingcore-spring.xml(your_custom_core-spring.xml)

```
<bean id="customerSiteAge"
class=“org.training.core.attributes.CustomerSiteAgeHandler"/>
```

Note:
customerSiteAge mentioned as spring bean id above should be same as attributeHandler mentioned in items.xml.

Step 5:
Build and then update the system either using HAC or using ant command

Now whenever we access customer Model, we can also access the customerSiteAgeattribute from customer model,
we will get the result of our custom logic in that attribute.

We can display it in any UI page by setting its value in appropriate controller and model attribute.

Advantages of Dynamic attribute
Data will not be saved in DB as its dynamic

The custom logic is written once and used all the time wherever that attribute is required.

When we should go for Dynamic attribute?

We should choose dynamic attribute whenever we want to get some derived data based on existing values.

So instead of saving one more column, we can make it as dynamic and compute its value based on the current values.

Dynamic in Enum?
Dynamic in enum is completely different from Dynamic attributes.

If an Enumtype is non-dynamic (by default, dynamic="false") we are not allowed to add new values at runtime.

If we add any non-dynamic enumtype without values,build will fail as it does not have any effect.

So if you want to add new values at runtime we have to make dynamic="true" for an enum.

We can change the flag anytime but enforces a system update.

If dynamic="false" the servicelayer generates real java enums (having a fixed set of values).

If dynamic="true" it generates hybris enums which can be used without fixed values(means we can add run time values).