Using Java reflection to reduce Code and Development time in DFS

Java reflections is one of the most powerful API's of Java Language, this can be used to reduce code significantly.

Most of the Current Enterprise application consists of different layers and they uses Value objects to transfer data from one layer to another. Inefficient way of using getters and setters of the attributes of Value objects can increase code and development time of application. Effective use of reflection can reduce code and development time significantly.

So lets take a Scenario, I have a Object type MyObjectType extending from dm_document with 50 additional attributes, so dm_document as of Documentum 6.5 has 86 attributes adding additional 50 attributes that means we have 139 attributes for this object type. Consider a standard Web Application using DFS behind which needs to manipulate (add or edit) instances of this object type, The Service needs to add all these attributes to the ***PropertySet*** of the ***DataObject*** representing that instance. Then need to call the appropriate service.

Considering that the bean instance name of MyObjectType is myObjectBean the Standard code will be something like this

```
ObjectIdentity objIdentity = new ObjectIdentity("myRepository");

DataObject dataObject = new DataObject(objIdentity, "dm_document");

PropertySet properties = dataObject.getProperties();

properties.set("object_name", myObjectBean.getObject_Name());

properties.set("title", myObjectBean.getTitle());

……
objectService.create(new DataPackage(dataObject), operationOptions);
```

In the above code you have to explicitly set individual attributes for the object , the more the number of attributes the more complex and messy code.

Take another Example, where you have to retrieve an Object information and pass it over to the UI layer.

```
myObjectBean.setObject_name(properties.get("object_name").getValueAsString());

myObjectBean.setTitle(properties.get("title").getValueAsString());

myObjectBean.setMy_Custom_Property(properties.get("my_custom_property").getValueAsString());
```

This operation can be more complex if you decide to use match the Data Type of your bean with the Object type.

So what is the best approach to reduce this complexity? the answer is effective use of reflection API.

Lets take a step to step approach to handle this issue.

To understand this better consider the below as the attributes of mycustomobjecttype

Attribute Name	Attribute Type
first_name	String
last_name	String
age	integer
date_purchased	time
amount_due	double
local_buyer	boolean

Java Bean

Create a Java Bean that matches the Object Type

```
public class Mycustomobjecttype {

  protected String first_name ;

  protected String last_name  ;

  protected int age;

  protected Date date_purchased  ;

  protected double amount_due  ;

  protected boolean local_buyer ;

  public int getAge() {

    return age;

  }

  public void setAge(int age) {

    this.age = age;

  }
```

```
  public double getAmount_due() {
    return amount_due;
  }
  public void setAmount_due(double amount_due) {
    this.amount_due = amount_due;
  }
  public Date getDate_purchased() {
    return date_purchased;
  }
  public void setDate_purchased(Date date_purchased) {
    this.date_purchased = date_purchased;
  }
  public String getFirst_name() {
    return first_name;
  }
  public void setFirst_name(String first_name) {
    this.first_name = first_name;
  }
  public String getLast_name() {
    return last_name;
  }
  public void setLast_name(String last_name) {
    this.last_name = last_name;
  }
  public boolean isLocal_buyer() {
    return local_buyer;
  }
  public void setLocal_buyer(boolean local_buyer) {
    this.local_buyer = local_buyer;
  }
}
```

Getting the Values from PropertySet (Loading Java Bean)

```
......

List<DataObject> dataObjectList = dataPackage.getDataObjects();

DataObject dObject = dataObjectList.get(0);

Mycustomobjecttype myCustomObject = new Mycustomobjecttype();

populateBeanFromPropertySet(dObject.getProperties(),myCustomObject);

......

// See the Reflection in Action here

public void populateBeanFromPropertySet(PropertySet propertySet, Object bean)
  throws Exception {

BeanInfo beaninformation;

beaninformation = Introspector.getBeanInfo(bean.getClass());

  PropertyDescriptor[] sourceDescriptors = beaninformation.getPropertyDescriptors();

  for (PropertyDescriptor descriptor : sourceDescriptors) {

    Object result = null;

    String name = descriptor.getName();

    if (!name.equals("class")) {

      if (propertySet.get(name) != null) {

        if (descriptor.getPropertyType().getName().equals(

            "int")) {

          result = new Integer(propertySet.get(name)

              .getValueAsString());

        } else if (descriptor.getPropertyType().getName().equals("double")) {

          result = new Double(propertySet.get(name).getValueAsString());

        } else if (descriptor.getPropertyType().getName().equals("boolean")) {

          result = new Boolean(propertySet.get(name).getValueAsString());
```

```
        } else if (descriptor.getPropertyType().getName().equals("java.util.Date")) {
          DateProperty dat = (DateProperty)propertySet.get(name);
          result = dat.getValue();
        }else {
          // none of the other possible types, so assume it as String
          result = propertySet.get(name).getValueAsString();
        }
        if (result != null)
          descriptor.getWriteMethod().invoke(bean, result);
      }
    }
  }
}
```

Setting Values to Property Set

```
public DataPackage createContentLessObject(Mycustomobjecttype myCustomType) throws Exception {
ObjectIdentity objectIdentity = new ObjectIdentity("testRepositoryName");
DataObject dataObject = new DataObject(objectIdentity, myCustomType.getClass().getName());

PropertySet properties = populateProperties(myCustomType);
properties.set("object_name",myCustomType.getFirst_name()+myCustomType.getLast_name() );
dataObject.setProperties(properties);

DataPackage dataPackage = new DataPackage(dataObject);
OperationOptions operationOptions = new OperationOptions();

return objectService.create(dataPackage, operationOptions);

}
```

```
// Reflection in Action

public PropertySet populateProperties(Object bean)throws Exception {

BeanInfo beaninfo;

PropertySet myPropertyset = new PropertySet();

  beaninfo = Introspector.getBeanInfo(bean.getClass());

  PropertyDescriptor[] sourceDescriptors = beaninfo

      .getPropertyDescriptors();

  for (PropertyDescriptor descriptor : sourceDescriptors) {

    String propertyName = descriptor.getName();

    if (!propertyName.equals("class")) {

        // dont set read only attributes if any

        // example r_object_id

      if (!propertyName.startsWith("r")) {

        Object value = descriptor.getReadMethod().invoke(bean);

        if (value != null) {

          myPropertyset.set(propertyName, value);

        }


      }


    }


  }

  return myPropertyset;

}
```