Assignment 3 Refactorings

1) The first bad code smell for my code I detected was a Long method, the way i fixed this bad code smell was by Extracting the code from the method and making a new method to handle it. This new method will be called in the old method in order to make the method smaller. This will give my code better readability and make it better structured.

Before Refactor:

```
if(Collection.class.isAssignableFrom(field.getType()))
{
    Object referenced = field.get(obj);
    int length = ((java.util.Collection<?>) referenced).size();
    objectElement.setAttribute(name:"length", Integer.toString(length));
    for(Object inArray : (java.util.Collection<?>) referenced)
    {
        uniqueID = id;
        idMap.put(inArray, uniqueID);
        Element fieldValue = new Element(name:"reference");
        fieldValue.addContent(Integer.toString(idMap.get(inArray)));
        objectElement.addContent(fieldValue);
        serialize(inArray);
    }
}       You, yesterday • implemented functionality for collection classes
```

After Refactor:

```
field.setAccessible(flag:true);
if(Collection.class.isAssignableFrom(field.getType()))
{
    collectionClass(field, obj, objectElement, uniqueID);
}
```

2) The second bad code smell for my code that I detected was the same code smell, the Long method. I fixed this bad code smell in the same way by removing the code into a new method to make my code better structurally and make it more readable.

Before Refactor:

```
else if(field.getType().isArray())
{
    if (field.getType().getComponentType().isPrimitive())
    {
        Object array = field.get(obj);
        int length = Array.getLength(array);
        objectElement.setAttribute(name:"length", Integer.toString(length));
        for(int i = 0; i < length; i++)
        {
            Object value = Array.get(array, i);
            Element arrayValue = new Element(name:"value");
            arrayValue.addContent(value.toString());
            objectElement.addContent(arrayValue);
        }
    }
    else
    {
        Object referenced = field.get(obj);
        int length = Array.getLength(referenced);
        objectElement.setAttribute(name:"length", Integer.toString(length));
        for(int i = 0; i < length; i++)
        {
            Object inArray = Array.get(referenced, i);
            uniqueID = id;
            idMap.put(inArray, uniqueID);
            Element fieldValue = new Element(name:"reference");
            fieldValue.addContent(Integer.toString(idMap.get(inArray)));
            objectElement.addContent(fieldValue);
            serialize(inArray);
        }
    }
}
```

After Refactor:

```
else if(field.getType().isArray())
{
    if (field.getType().getComponentType().isPrimitive())
    {
        primitiveArrayMethod(field, obj, objectElement);
    }
    else
    {
        objectArrayMethod(field, obj, objectElement, uniqueID);
    }
}
```

3) The third refactor is also from the same bad code smell, the Long Method. I fixed this the same way as the other methods by extracting the code and putting it into a helper method. This makes my code better readable and better structurally.

Before Refactor:

```
    else if(!field.getType().isPrimitive())
    {
        Element fieldElement = new Element(name:"field");
        fieldElement.setAttribute(name:"name", field.getName());
        fieldElement.setAttribute(name:"declaringclass", className);
        objectElement.addContent(fieldElement);
        Object referenced = field.get(obj);
        if(!idMap.containsKey(referenced))
        {
            uniqueID = id;
            idMap.put(referenced, uniqueID);
            Element fieldValue = new Element(name:"reference");
            fieldValue.addContent(Integer.toString(idMap.get(referenced)));
            fieldElement.addContent(fieldValue);
            serialize(referenced);
        }
        else
        {
            Element fieldValue = new Element(name:"reference");
            fieldValue.addContent(Integer.toString(idMap.get(referenced)));
            fieldElement.addContent(fieldValue);
        }
    }
    else
    {
        Element fieldElement = new Element(name:"field");
        fieldElement.setAttribute(name:"name", field.getName());
        fieldElement.setAttribute(name:"declaringclass", className);
        objectElement.addContent(fieldElement);
        Element fieldValue = new Element(name:"value");
        fieldValue.addContent(field.get(obj).toString());
        fieldElement.addContent(fieldValue);
    }
}
```

After Refactor:

```
else if(!field.getType().isPrimitive())
{
  objectFieldMethod(objectElement, field, obj, className, uniqueID);
}
else
{
   primitiveFieldMethod(objectElement, field, obj, className);
}       You, 2 days ago • integrated functionailty to allow objects refe
```