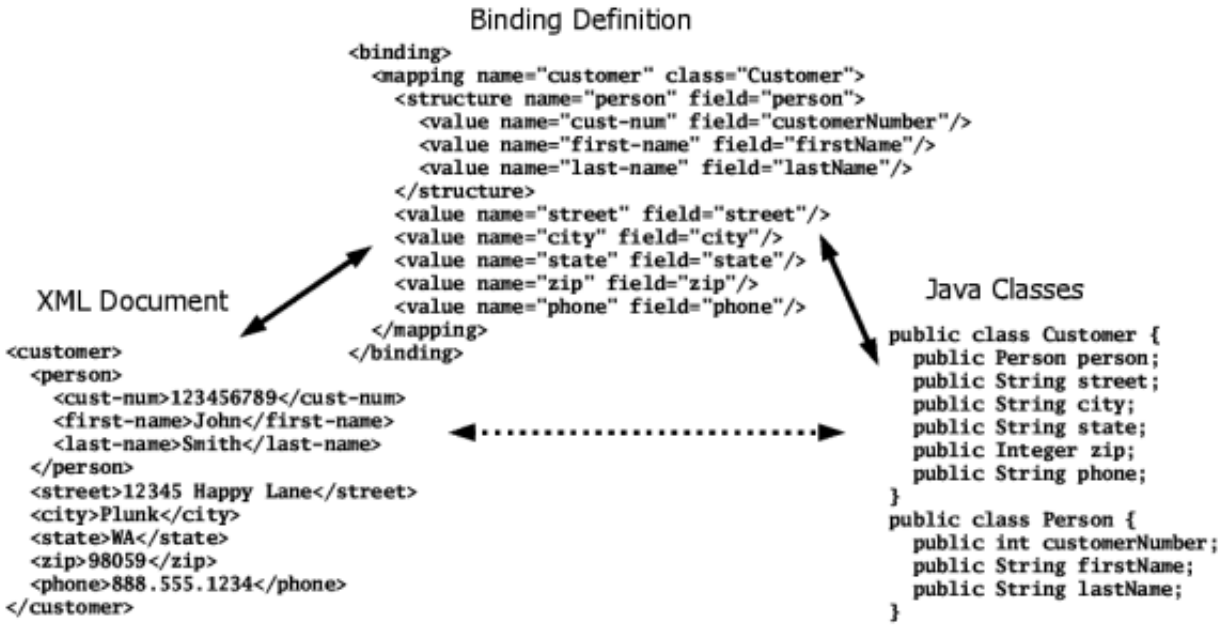
Structure mapping

JiBX's binding definition file provides great flexibility for converting between XML documents and Java object structures. Other data binding frameworks generally limit customization to just the names and the style (attribute or element) of XML representations. JiBX goes further by allowing ***structure mapping*** between XML and object structures.

Figure 4 shows the same example as Figure 1. This basic type of binding, where the structure of the XML is the same as the structure of the Java objects, is typical of what's supported by most data binding frameworks.

Figure 4. Simple binding example



What most frameworks will ***not*** allow you to do is to bring the values within the **name** element into the the object that corresponds to the **customer** element. In other words, you can't map the XML to this class:

```
public class Customer {
  public int customerNumber;
  public String firstName;
  public String lastName;
  public String street;
  public String city;
  public String state;
```

```
    public Integer zip;
    public String phone;
  }
```

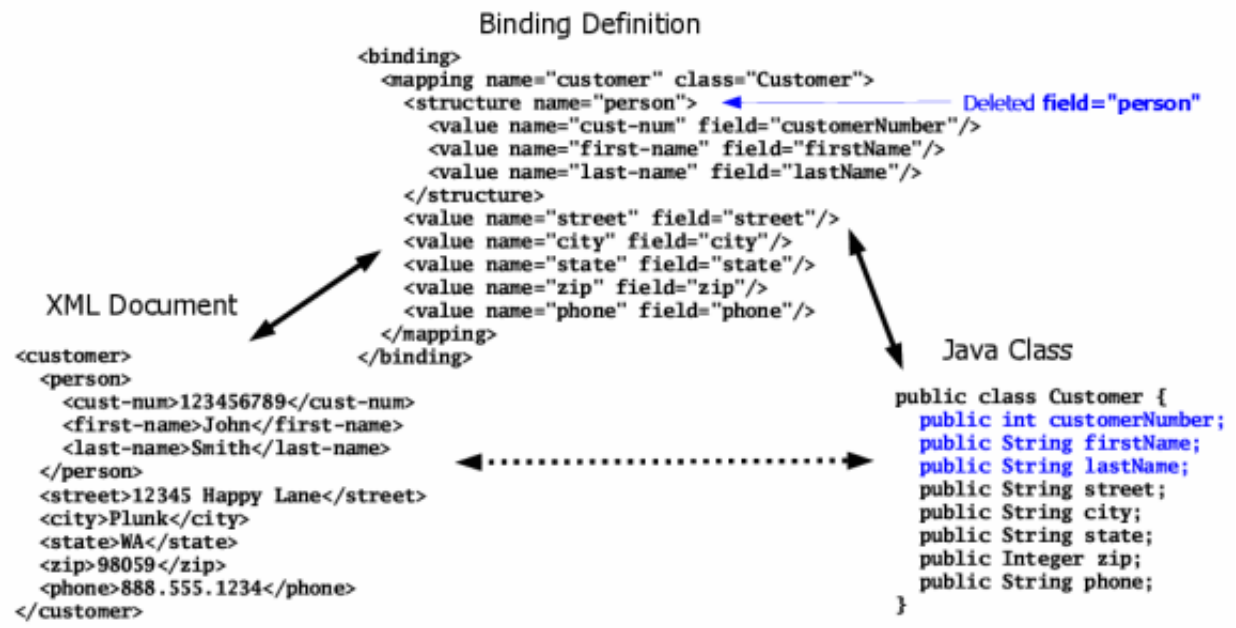
Nor can you map the XML to a pair of classes like these:

```
public class Customer {
  public int customerNumber;
  public String firstName;
  public String lastName;
  public Address address;
  public String phone;
}

public class Address {
  public String street1;
  public String city;
  public String state;
  public String zip;
}
```

JiBX ***does*** allow you to do this type of mapping. This means the structure of your objects is not tied to the structure of the XML - you can restructure your object classes without needing to change the XML format used for external data transfer. Figure 5 shows a binding definition for the first case above, with the changes shown in blue.

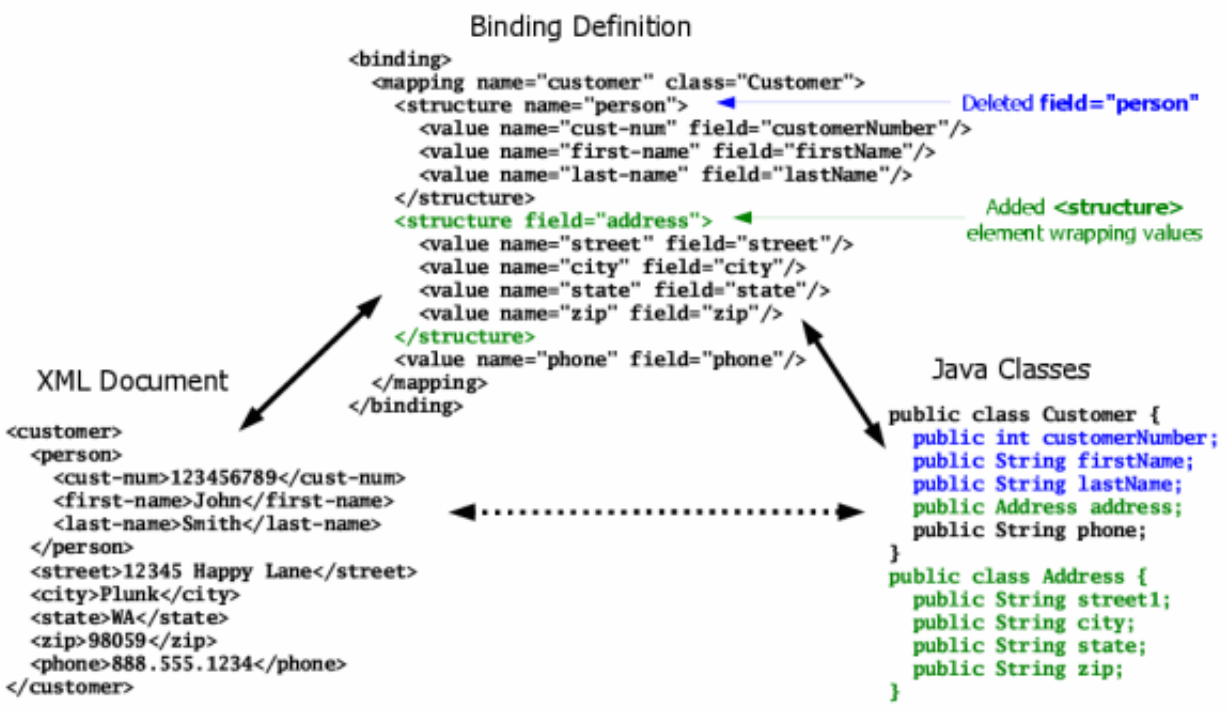
Figure 5. Flattened binding



The only difference in this binding definition from Figure 4 is that I've removed the **field** attribute from the **structure** element. The structure still defines an XML element name, but without the **field** attribute the values within that XML element are treated as properties of the containing object (the `Customer` class instance).

Adding a new **structure** element to the binding definition handles the second structure change case, as shown in Figure 6. I've highlighted the changes from Figure 4 in green.

Figure 6. Split binding

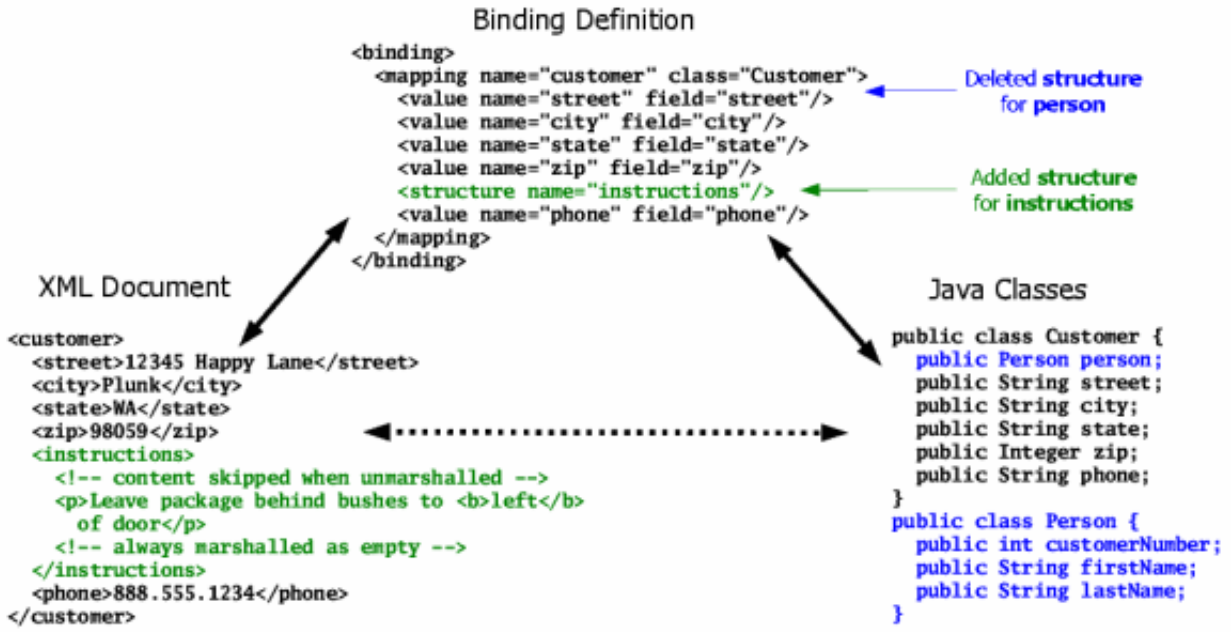


Here the added **structure** element uses a **field** attribute but no **name** attribute. This tells JiBX that the properties from the `Address` class instance referenced by the `address` field should be included directly as children of the **customer** element in the XML document.

As a final example of structure mapping, Figure 7 demonstrates partial mappings, where not all the components of the XML document or the Java classes are actually included in the binding. On the Java side, all that's needed is to leave something out of the binding defintion, as shown highlighted in blue for the `Person` class. On the XML side, the green highlighted portions of the diagram show how you can use a **structure** element in the binding definition with a **name** attribute but no property or child definitions. This tells JiBX to require an element with that name but ignore any content when unmarshalling. When marshalling, the element will always be generated as empty. If the **structure** is optional rather than required,

the corresponding element will be ignored (if present) when unmarshalling and will be skipped completely when marshalling. See the <structure> element details page for more information on the different usages of the **structure** element.

Figure 7. Ignored components



The flexibility provided by structure mapping allows you to decouple your XML data representations from your actual Java class structure. It lets you interface existing code to new XML representations, and also lets you preserve an existing XML representation while refactoring your code to better represent the use of data within your application. This decoupling of XML structure from code structure is one of the most powerful features of the JiBX framework.

Next: Working with collections and IDs