# Table of Contents

- Defining Metadata for classes ........................................ 2
- Accessing Metadata for classes ..................................... 4
- MetaData Reference ....................................................... 5
  - JDOMetadata ......................................................... 5
  - PackageMetadata ................................................... 5
  - ClassMetadata ...................................................... 5
  - InterfaceMetadata .................................................. 5
  - FieldMetadata ....................................................... 5
  - PropertyMetadata ................................................... 5
  - CollectionMetadata ................................................ 5
  - ArrayMetadata ........................................................ 5
  - MapMetadata .......................................................... 6
  - ElementMetadata ..................................................... 6
  - KeyMetadata ........................................................... 6
  - ValueMetadata ........................................................ 6
The JDO API provides a dynamic API for defining metadata for classes, as an alternative to using annotations or XML metadata.
Defining Metadata for classes

The basic idea behind the Metadata API is that the developer obtains a metadata object from the `PersistenceManagerFactory`, and adds the definition to that as required, before registering it for use in the persistence process.

```java
PersistenceManagerFactory pmf = JDOHelper.getPersistenceManagerFactory(propsFile);
...
JDOMetadata md = pmf.newMetadata();
```

So we have a `JDOMetadata` object and want to define the persistence for our class `mydomain.MyClass`, so we do as follows

```java
PackageMetadata pmd = md.newPackageMetadata("mydomain");
ClassMetadata cmd = pmd.newClassMetadata("MyClass");
```

So we follow the same structure of the JDO XML Metadata file adding packages to the top level, and classes to the respective package. Note that we could have achieved this by a simple typesafe invocation

```java
ClassMetadata cmd = md.newClassMetadata(MyClass.class);
```

So now we have the class defined, we need to set its key information

```java
cmd.setTable("CLIENT").setDetachable(true).setIdentityType(IdentityType.DATASTORE);
cmd.setPersistenceModifier(ClassPersistenceModifier.PERSISTENCE_CAPABLE);

InheritanceMetadata inhmmd = cmd.newInheritanceMetadata();
inhmmd.setStrategy(InheritanceStrategy.NEW_TABLE);
DiscriminatorMetadata dmd = inhmmd.newDiscriminatorMetadata();
dmd.setColumn("disc").setValue("Client");
dmd.setStrategy(DiscriminatorStrategy.VALUE_MAP).setIndexed(Indexed.TRUE);

VersionMetadata vermd = cmd.newVersionMetadata();
vermd.setStrategy(VersionStrategy.VERSION_NUMBER);
vermd.setColumn("version").setIndexed(Indexed.TRUE);
```

And we define also define fields/properties via the API in a similar way

```java
FieldMetadata fmd = cmd.newFieldMetadata("name");
fmd.setNullValue(NullValue.DEFAULT).setColumn("client_name");
fmd.setIndexed(true).setUnique(true);
```

Note that, just like with XML metadata, we don't need to add information for all fields since they
have their own default persistence settings based on the type of the field.

As you can see from the objects in this API, it follows the exact same structure as the JDO XML metadata, so you should be able to specify all by working your way through the respective javadocs for the API classes.

All that remains is to register the defined metadata with the persistence process

```java
pmf.registerMetadata(md);
```
Accessing Metadata for classes

Maybe you have a class with its persistence defined in XML or annotations and you want to check its persistence information at runtime. With the JDO Metadata API you can do that

```java
TypeMetadata compmd = pmf.getMetadata("mydomain.MyOtherClass");
```

and we can now inspect the information, casting the `compmd` to either `javax.jdo.metadata.ClassMetadata` or `javax.jdo.metadata.InterfaceMetadata`.

ℹ️ you cannot currently change metadata retrieved in this way, only view it.
MetaData Reference

JDOMetadata
This represents a JDO context (the equivalent of a package.jdo file), and contains packages, named queries, fetch plans etc. [javadoc]

PackageMetadata
This represents a package (the package in package.jdo), and contains classes/interfaces, sequences etc. [javadoc]

ClassMetadata
This represents a persistable class (the class in package.jdo), and contains members (fields/properties), identity, version, indices, unique constraints, FKs, named queries, fetch groups for the class, and unmapped columns. [javadoc]

InterfaceMetadata
This represents a persistable interface (the interface in package.jdo), and contains members (fields/properties), identity, version, indices, unique constraints, FKs, named queries, fetch groups for the class, and unmapped columns. [javadoc]

FieldMetadata
This represents a persistable field (the field in package.jdo), and contains settings for the field, array/collection/map mapping, as well as any converters, columns etc. [javadoc]

PropertyMetadata
This represents a persistable property (the property in package.jdo), and contains settings for the property, array/collection/map mapping, as well as any converters, columns etc. [javadoc]

CollectionMetadata
This represents a collection member (the collection in package.jdo), and contains settings for the collection elements. [javadoc]

ArrayMetadata
This represents an array member (the array in package.jdo), and contains settings for the array elements. [javadoc]
MapMetadata

This represents a map member (the map in package.jdo), and contains settings for the map keys/values.

ElementMetadata

This represents a collection/array element (the element in package.jdo), and contains settings for the collection/array element.

KeyMetadata

This represents a map key (the key in package.jdo), and contains settings for the map key.

ValueMetadata

This represents a map value (the element in package.jdo), and contains settings for the map value.