Creating an Object Metamodel Using Annotations and Reflection

José Paumard
PHD, JAVA CHAMPION, JAVA ROCK STAR

@JosePaumard https://github.com/JosePaumard

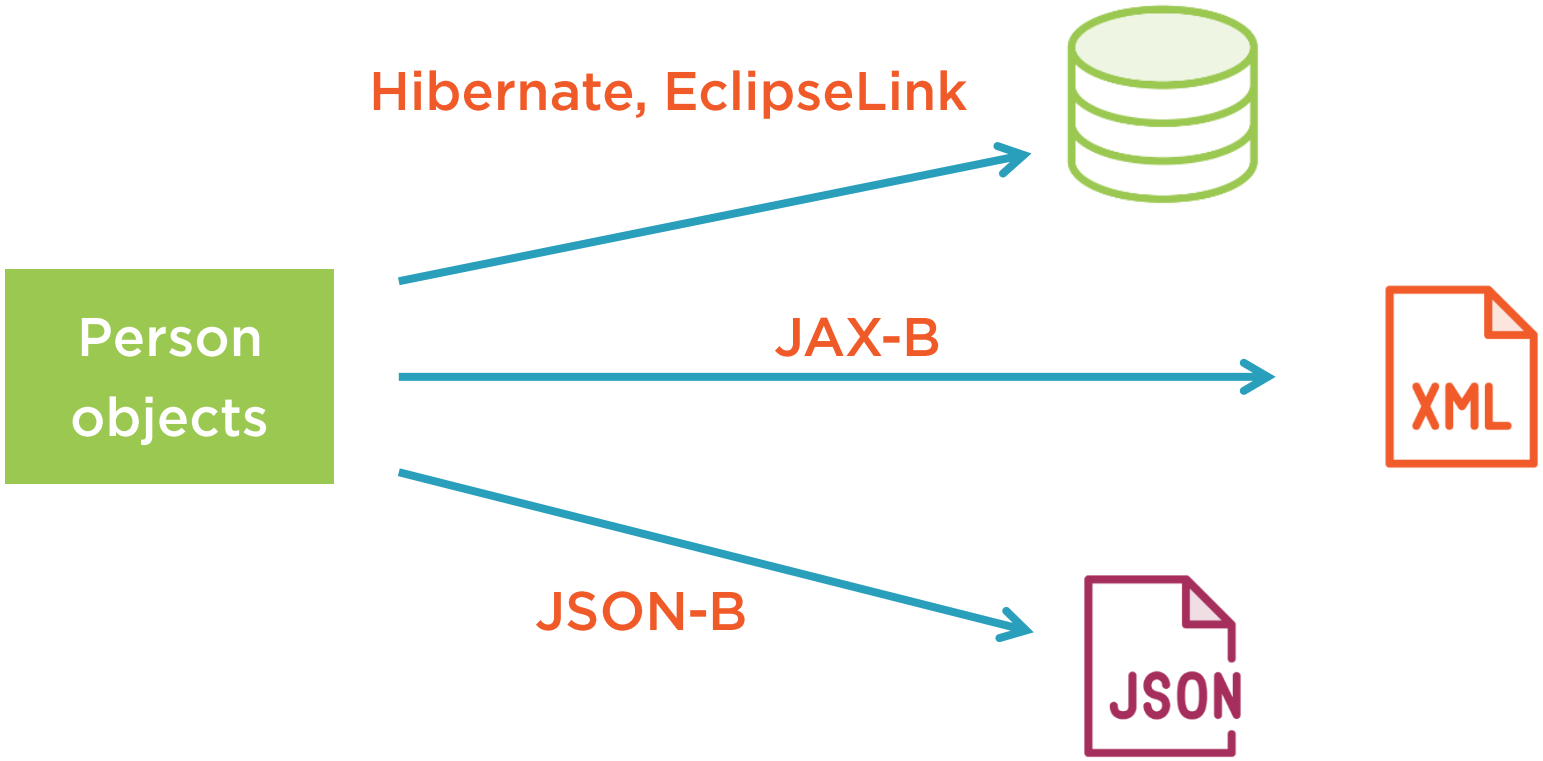
Agenda

Create a class

Add annotations to its elements

The goal is to use this metadata for other things

Mapping Objects to Files and Databases



Creating a Metamodel

What we need is to tell what to save from the objects

And how to save it

What we need is metadata added to the Person class

```java
public class Person {
    @PrimaryKey
    private long id;
    @Column
    private int age;
    @Column
    private String name;

    // getters and setters
}
```

```java
public @interface PrimaryKey {
}

public @interface Column {
}
```

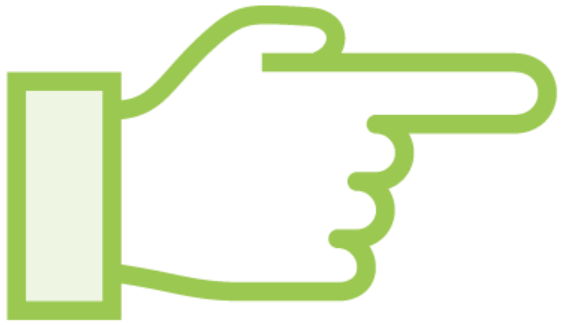
Here is our Person class

We need to tell that id is a primary key

And that age and name are regular fields

Creating custom annotations is so easy!

Getting the Annotations of an Element



An annotation adds information to an element in a class

- the class itself

- a field

- a constructor

- a method

- a parameter in a method or a constructor

It can be read by reflection

```
public interface AnnotatedElement {

  boolean isAnnotationPresent(Class type);

  Annotation getAnnotation(Class type);
  Annotation[] getAnnotations();
}
```

Checks if a given annotation is present

Get the annotations given its type

Annotations can be read on anything that implements AnnotatedElement

Using Reflection to Read and Write a Field

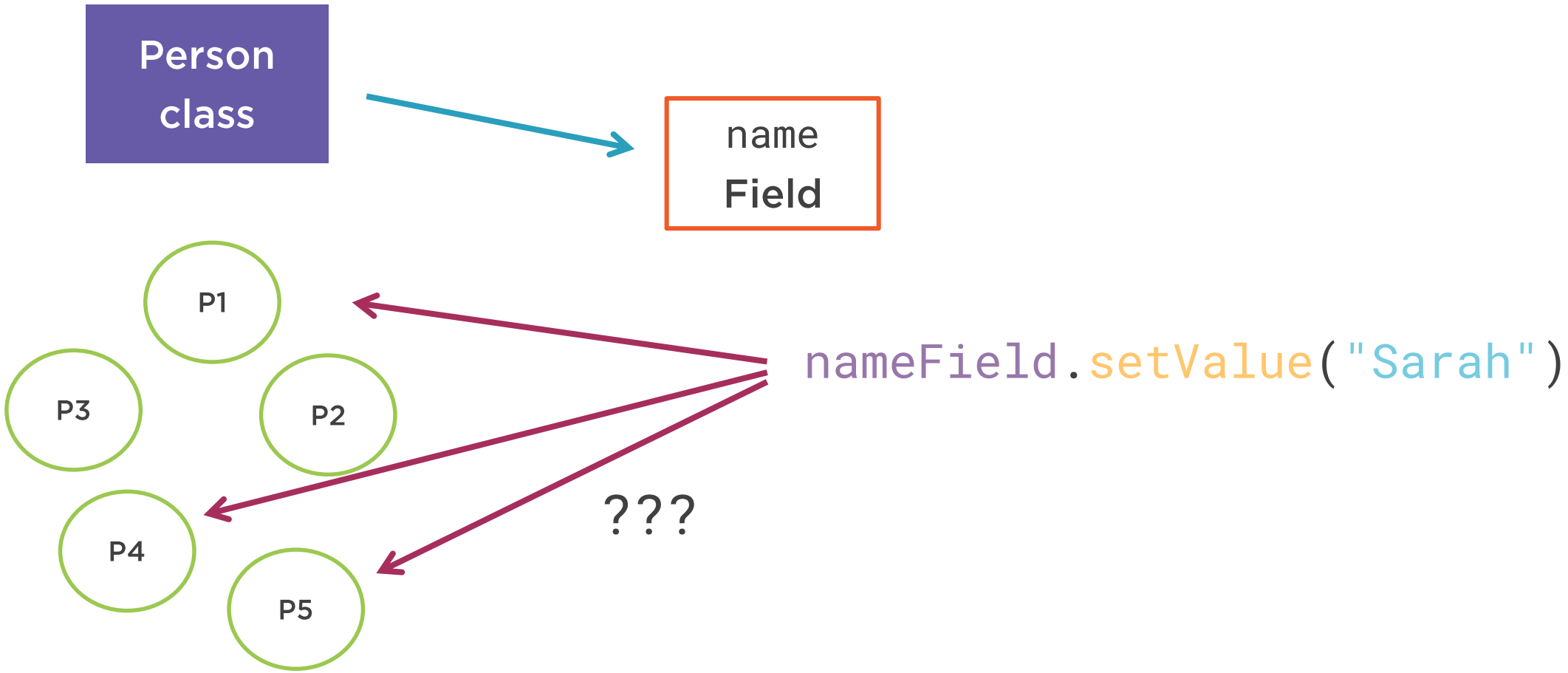
The problem is the following:

Reading a person bean

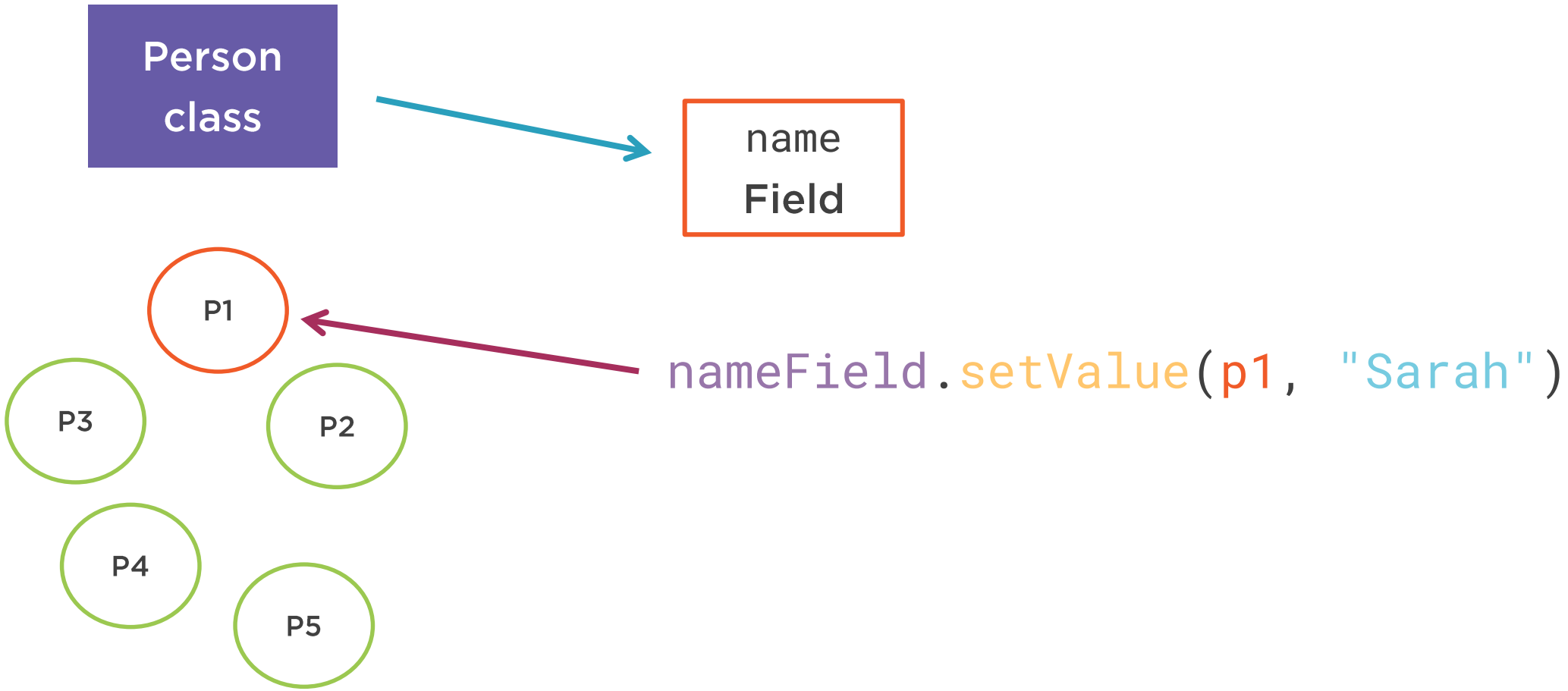
To write its content in a storage

How to read fields with reflection?

Setting the Value of a Field



Setting the Value of a Field



```
Person o = ...;
Class<?> clss = o.getClass();
Field field = clss.getDeclaredField("name");


field.setValue(o, "Sarah");
```

The pattern is to set the value of a field for an object to a given value

What about private fields?

Is this API breaking encapsulation?

If the given field is private then an IllegalAccessException is thrown

But there is the setAccessible(true) call

It does not make a private field public!

It just suppresses the access control on that field

```
Person o = ...;
Class<?> clss = o.getClass();
Field field = clss.getDeclaredField("name");

field.setAccessible(true);
field.setValue(o, "Sarah");
```

If the field name is private, then call setAccessible(true)

```
Person o = ...;
Class<?> clss = o.getClass();
Field field = clss.getDeclaredField("name");

field.setAccessible(true);
String name = (String)field.getValue(o);
```

And the same goes for the read() pattern

Writing the Person Beans to a Storage

```java
public interface EntityManager<T> {

    void persist(T t);

    T read(Class<?> clzz, long primaryKey);
}
```

The EntityManager interface models the writing and the reading of instances of T to any storage file or media

Without knowing what is T at compile time

Given an instance of T

1) read the fields of T

2) check for the annotations

3) find the primary key

4) find the fields to read / write

```
Class<?> clss = t.getClass();
Field[] fields = clss.getDeclaredFields();

for (Field field: fields) {
   if (field.getAnnotation(PrimaryKey.class) != null) {
      // this is the primary key
   }
   if (field.getAnnotation(Column.class) != null) {
      // this is an element to read / write
   }
}
```

First, we need to loop through the fields of the class

Then check if the @PrimaryKey annotation is present

And the same for the @Column annotation

Demo

Let us see some code!

Let us add annotations to a Person bean

Use them to read its fields

Module Wrap Up

What did you learn?

How to add annotations to fields

How to read and write fields using the Reflection API

What about writing objects to a database?