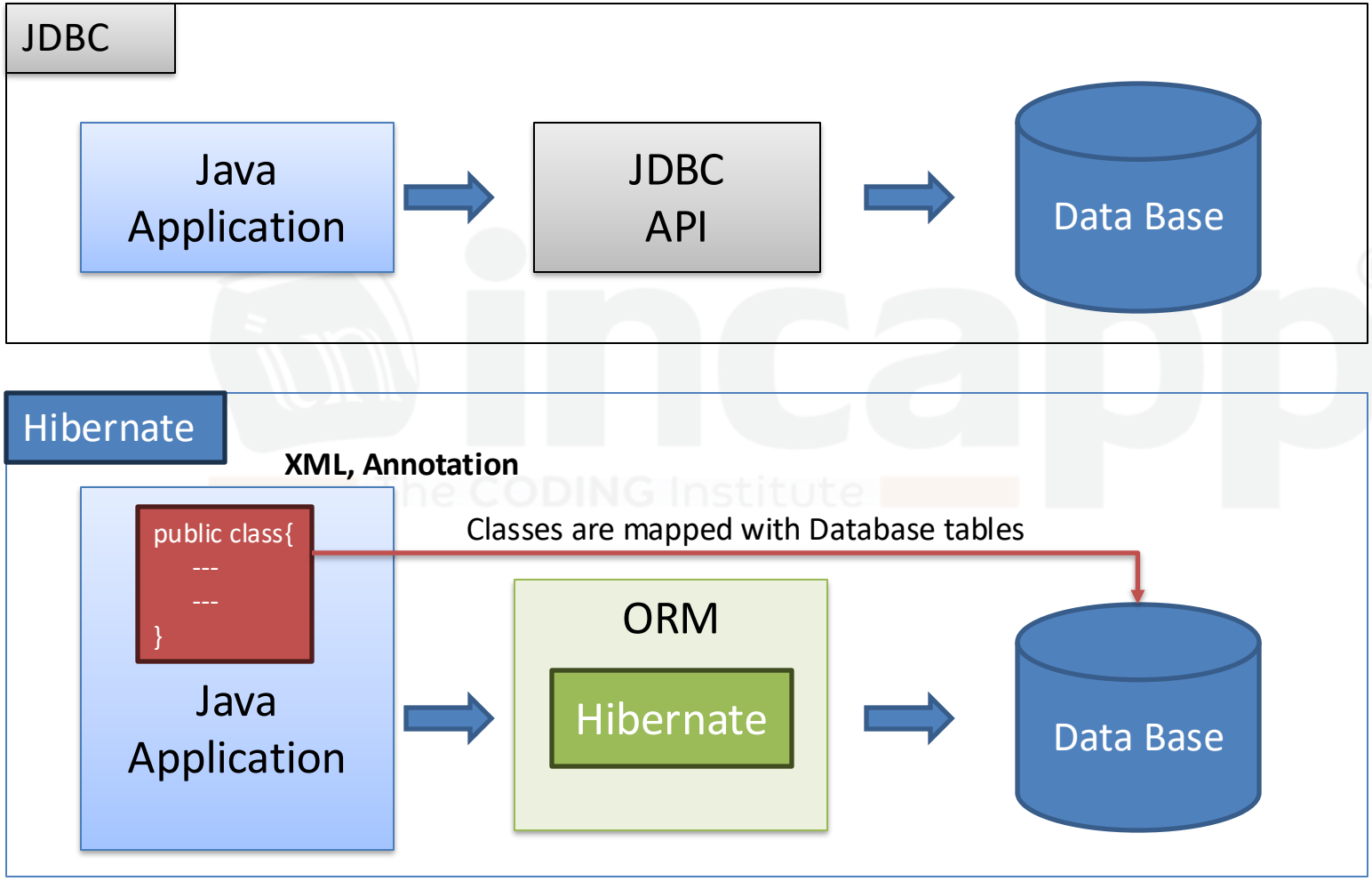
Hibernate Framework

Hibernate ORM (Object-Relational Mapping) is an open-source Java framework that simplifies the development of Java applications interacting with a database. It provides a framework for mapping an object-oriented domain model to a relational database and automates the tedious task of database operations.

- Hibernate was developed by **Gavin King** with colleagues from Cirrus Technologies in 2001.
- Hibernate is latter maintained by JBoss, Inc. (now part of **Red Hat**).
- Hibernate introduced as an alternative to using EJB2 entity beans.
- Hibernate implements **JPA** (formerly Java Persistence API, now Jakarta Persistence API).
- Hibernate is a **non-inversive** framework, means it won't forces the programmers to extend/implement any class/interface.

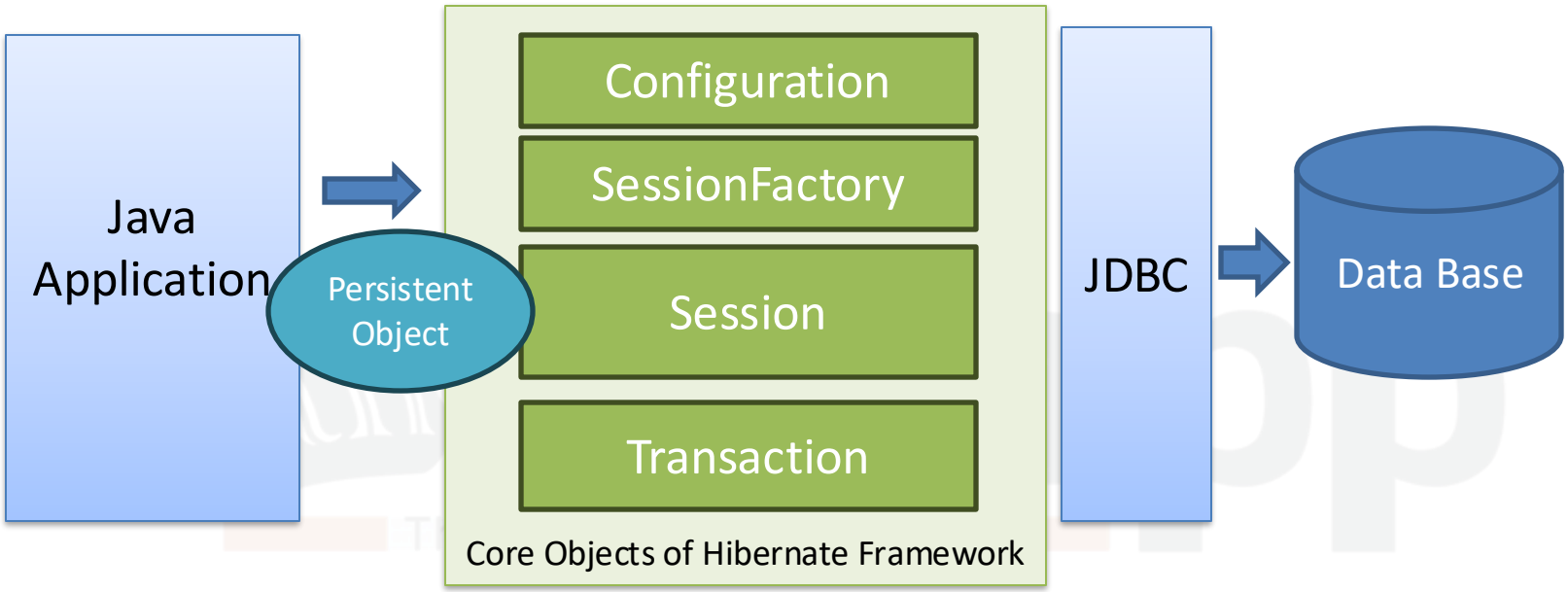
JDBC vs Hibernate



Hibernate Features

1. **Object-Relational Mapping**: Maps Java classes to database tables.
2. **HQL (Hibernate Query Language)**: A query language similar to SQL but operates on objects instead of tables.
3. **Lazy Loading**: Loads data only when it's accessed, improving performance.
4. **Cache Mechanism**: Supports first-level and second-level caching for better performance.
5. **Automatic Table Schema Generation**: Can create or update database schemas based on object mappings.
6. **Cross-Database Compatibility**: Database-independent via Dialects.

Hibernate Architecture



Configuration: Reads the configuration file (***hibernate.cfg.xml***) for database and mapping details.
SessionFactory: A heavyweight object that creates Session instances for database operations.
Session: A lightweight, short-lived object representing a unit of work.
Transaction: Abstracts the database transaction.
Query: Allows HQL or native SQL execution.

hibernate.cfg.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
        <session-factory>
        <property name="connection.driver_class">com.mysql.cj.jdbc.Driver</property>
        <property name= "connection.url">jdbc:mysql://localhost:3306/hibernate_db</property>
        <property name="connection.username">root</property>
        <property name="connection.password">Incapp@12</property>
        <property name="dialect">org.hibernate.dialect.MySQL8Dialect</property>
        <!-- Create new Tables for everytime -->
        <!-- <property name= "hbm2ddl.auto">create</property> -->
        <!-- Create new Tables if does not exist. -->
        <property name= "hbm2ddl.auto">update</property>
        <!-- To show sql query on cosole -->
        <property name="show_sql">true</property>
        <!-- To show sql query in formatted way -->
        <property name="format_sql">true</property>
        </session-factory>
</hibernate-configuration>
```

Book Entity for Database table

```java
package com.incapp;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
public class Book {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int id;
	private String name;
	private int price;
	private String author;

	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
```

continue…

Book Entity for Database table

```
	public void setName(String name) {
		this.name = name;
	}
	public int getPrice() {
		return price;
	}
	public void setPrice(int price) {
		this.price = price;
	}
	public String getAuthor() {
		return author;
	}
	public void setAuthor(String author) {
		this.author = author;
	}

	@Override
	public String toString() {
		return "Book [id=" + id + ", name=" + name + ", price=" + price + ", author=" + author + "]";
	}
}
```

Getting Session and Transaction objects

```
Configuration cfg=new Configuration();
cfg.configure(); //Read the hibernate.cfg.xml file automatically
// cfg.configure("abc.xml"); //Can give your own name
cfg.addAnnotatedClass(Book.class); //Create a Book table in Database
SessionFactory factory=cfg.buildSessionFactory(); //Connected with data base
Session session= factory.openSession();
Transaction txn=ses.beginTransaction();
```

Insert a row in DB

```
//Create Book Object to insert in DB
Book b=new Book();
b.setName("Java");
b.setPrice(900);
b.setAuthor("Rahul Chauhan");

session.save(b); //Insert Book data inside the Book Table of DB
txn.commit(); //Commit to confirm the insertion in DB

session.close();
```

Read Data by ID

```
Session session= factory.openSession();
Book b= session.get(Book.class, 1);
System.out.println(b);
```

Hibernate Annotations

Annotations	Details
@Entity	Create table with class name
@Table	Create table with own choice name
@Id	Create primary key (PK)
@GeneratedValue	Auto Increment for PK
@Column	Own choice Column name or limit length
@Transient	Ignore the column
@Temporal	Specifies the Date format
@Lob	Large size data like blob data (image, file etc.)
@OneToOne	OneToOne mapping
@OneToMany	OneToMany mapping
@ManyToOne	ManyToOne mapping
@ManyToMany	ManyToMany mapping

Update Data

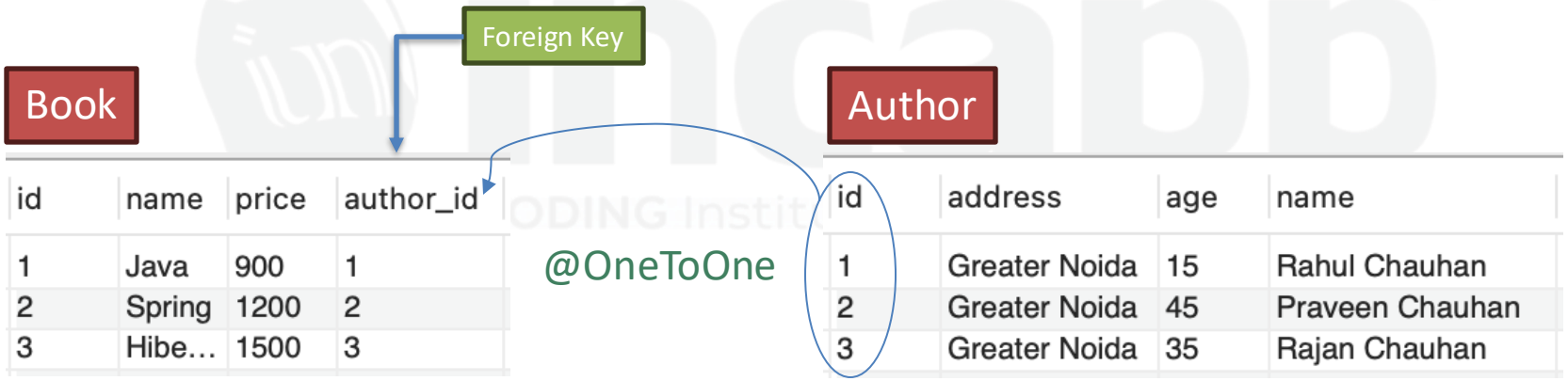
```
Book b=session.get(Book.class, 1); //get Book by ID
b.setName("Spring");
b.setAuthor("INCAPP");
b.setPrice(2000);
//b.setId(102); //can not update ID, because it is primary key

session.saveOrUpdate(b);
txn.commit();
session.close();
```

Delete Data

```
Book b=ses.get(Book.class, 1); //If Book not found, java.lang.IllegalArgumentException
ses.delete(s1);
txn.commit();
session.close();
```

One To One Mapping

Use @OneToOne in Book class to link Book table with Author table.
One book mapped with one author only.
Can fetch author via book, but can not get book via author.
@OneToOne is **unidirectional** by default.



id	name	price	author_id
1	Java	900	1
2	Spring	1200	2
3	Hibe…	1500	3

id	address	age	name
1	Greater Noida	15	Rahul Chauhan
2	Greater Noida	45	Praveen Chauhan
3	Greater Noida	35	Rajan Chauhan

One To One Uni-Directional

```java
@Entity
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String name;
    private int price;

    @OneToOne
    private Author author;
```

@OneToOne **Uni-Directional**

```java
@Entity
public class Author {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private int age;
    private String name;
    private String address;
```

One To One Bi-Directional

For **bidirectional**, use @OneToOne in Author class also.
Now, can fetch data from author from book and book from author also.

```java
@Entity
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String name;
    private int price;

    @OneToOne
    private Author author;
```

@OneToOne **Bi-Directional**

```java
@Entity
public class Author {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private int age;
    private String name;
    private String address;

    @OneToOne(mappedBy = "author")
    private Book book;
```