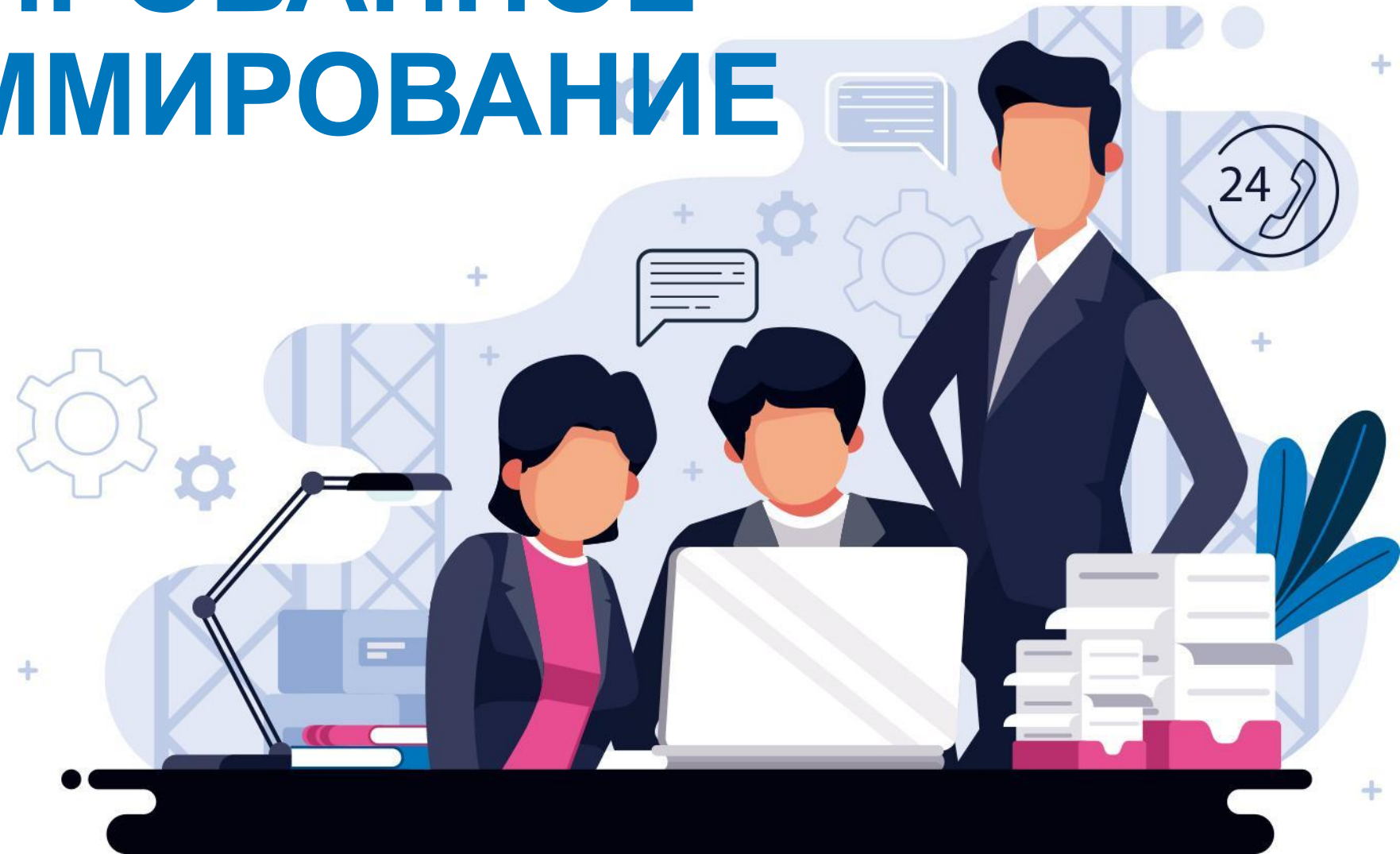


ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Лекция 6



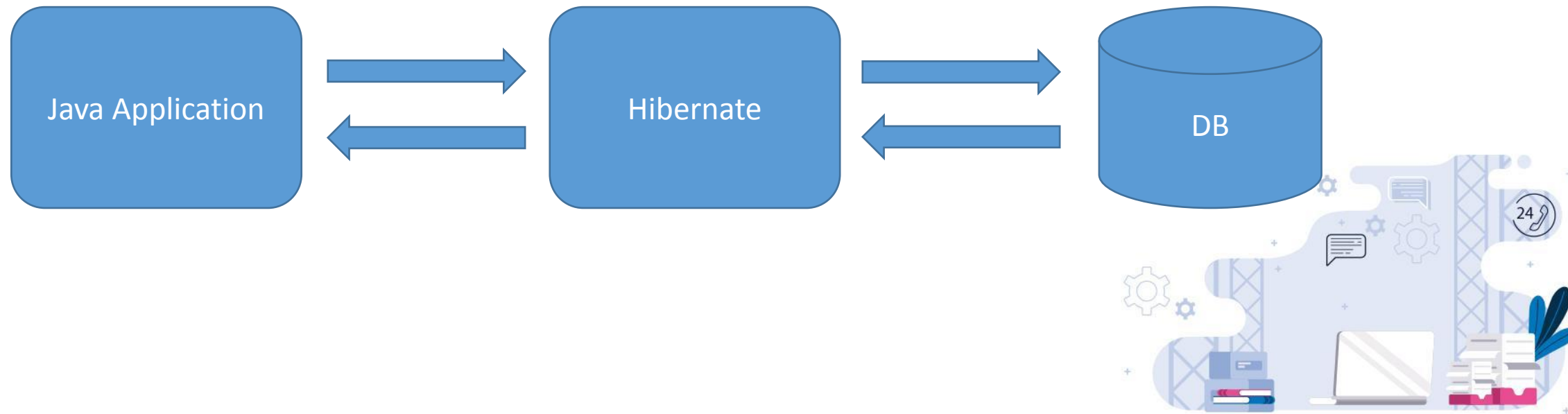
План

- Hibernate
- Подключение Hibernate
- Подключение Java приложения к БД
- Создание связи между классом и таблицей
- Сохранение Java объектов в БД
- Генерация значений для столбца Primary Key
- Получение Java объектов из БД
- Изменение Java объектов в БД
- Удаление Java объектов из БД



Hibernate

Hibernate – это ORM фреймворк, который используется для создания, чтения, редактирования и удаления записей из базы данных (CRUD).



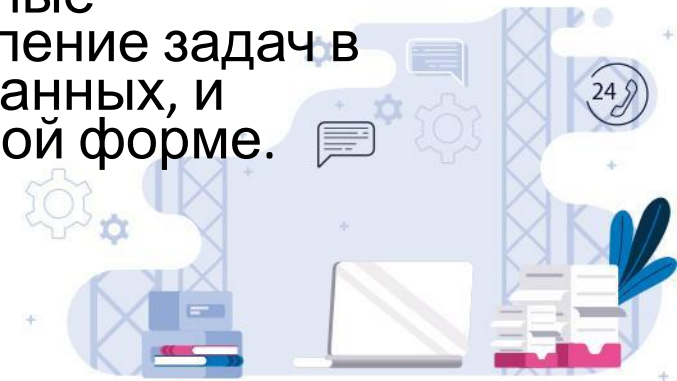
Hibernate

Hibernate предоставляет технологию ORM.

ORM или Object-relational mapping (рус. Объектно-реляционное отображение) — это технология программирования, которая позволяет преобразовывать несовместимые типы моделей в ООП, в частности, между хранилищем данных и объектами программирования.

ORM используется для упрощения процесса сохранения объектов в реляционную базу данных и их извлечения, при этом ORM сама заботится о преобразовании данных между двумя несовместимыми состояниями.

Большинство ORM-инструментов в значительной мере полагаются на метаданные базы данных и объектов, так что объектам ничего не нужно знать о структуре базы данных, а базе данных — ничего о том, как данные организованы в приложении. ORM обеспечивает полное разделение задач в хорошо спроектированных приложениях, при котором и база данных, и приложение могут работать с данными каждый в своей исходной форме.

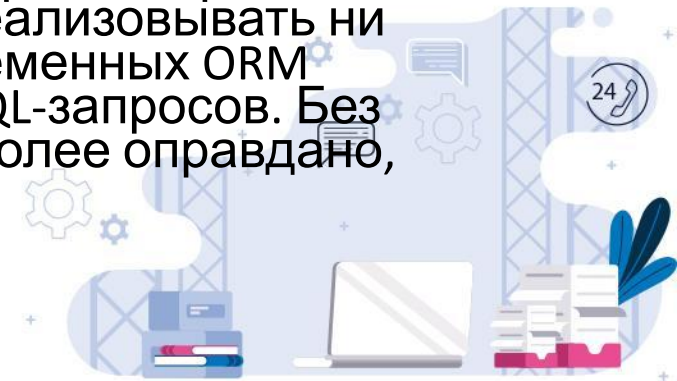


Hibernate

Преимущества и недостатки ORM

Использование ORM в проекте избавляет разработчика от необходимости работы с SQL и написания большого количества кода, часто однообразного и подверженного ошибкам. Весь генерируемый ORM код предположительно хорошо проверен и оптимизирован, поэтому не нужно в целом задумываться о его тестировании. Это несомненно является плюсом.

Основной минус — это потеря производительности. Это происходит потому, что большинство ORM предназначены для обработки широкого спектра сценариев использования данных, гораздо большего, чем любое отдельное приложение когда-либо сможет использовать. Вопрос о целесообразности использования ORM по большому счету затрагивается только в больших проектах, которые сталкиваются с высокой нагрузкой, здесь приходится выбирать что более приоритетно — удобство или производительность? Конечно, работа с БД посредством грамотно написанного SQL-кода будет намного эффективнее, но не стоит забывать и о таком параметре, как время — то, что с легкостью пишется с использованием ORM за неделю, можно реализовывать ни один месяц собственными усилиями. Кроме того, большинство современных ORM позволяют программисту при необходимости самому задавать код SQL-запросов. Без сомнений, для небольших проектов использование ORM будет куда более оправдано, чем разработка собственных библиотек для работы с БД.



Hibernate

Hibernate регулирует SQL-запросы. Например, при вызове метода **save** происходит:

- сбор данных из полей объекта;
- формирование INSERT выражения для добавления новой строки в таблицу.

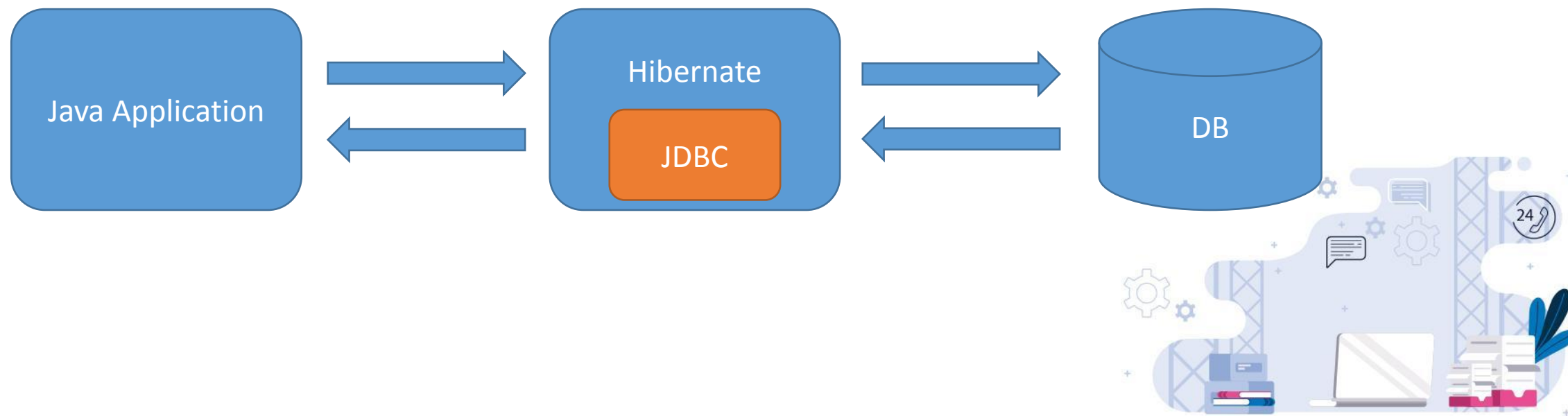
При вызове метода **get** происходит:

- формирование SELECT выражения для получения необходимых данных;
- создание объекта Java класса и присвоение его полям значений, полученных из базы.



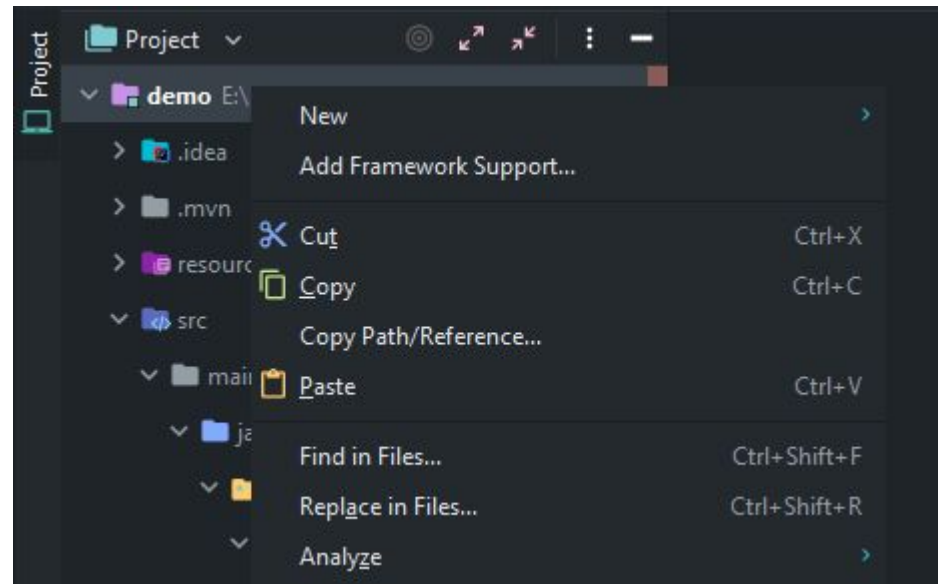
Hibernate

Использование **Hibernate** позволяет сократить количество кода (в сравнении, например, с JDBC). На самом деле Hibernate сам использует JDBC, являясь таким образом посредником.



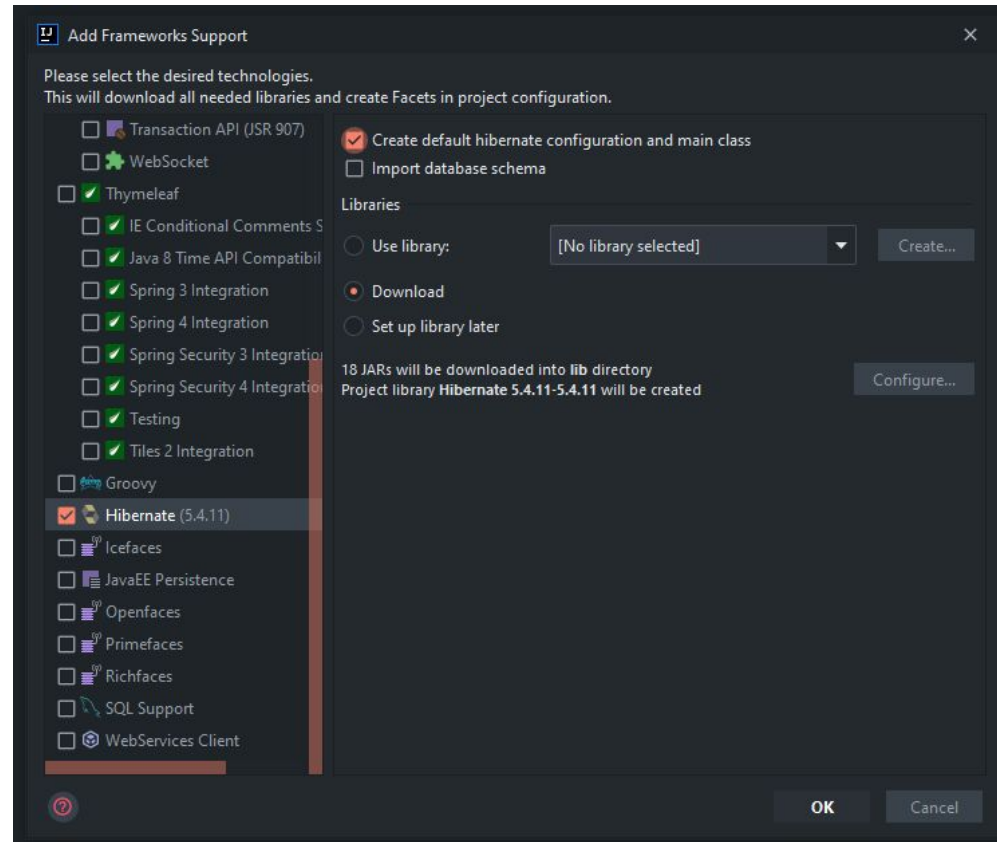
Подключение Hibernate

Прежде всего необходимо подключить Hibernate к проекту. Для этого необходимо нажать на папку проекта правой кнопкой мыши и выбрать Add Framework Support.



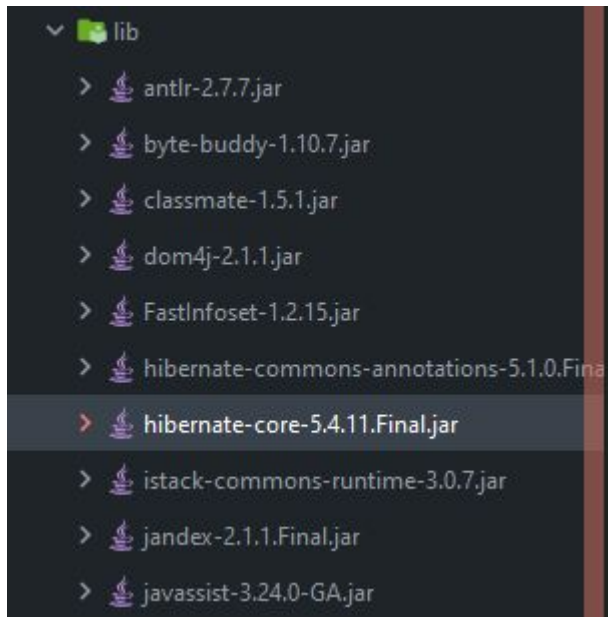
Подключение Hibernate

В списке необходимо выбрать Hibernate и поставить галочку напротив Create default hibernate configuration and main class.



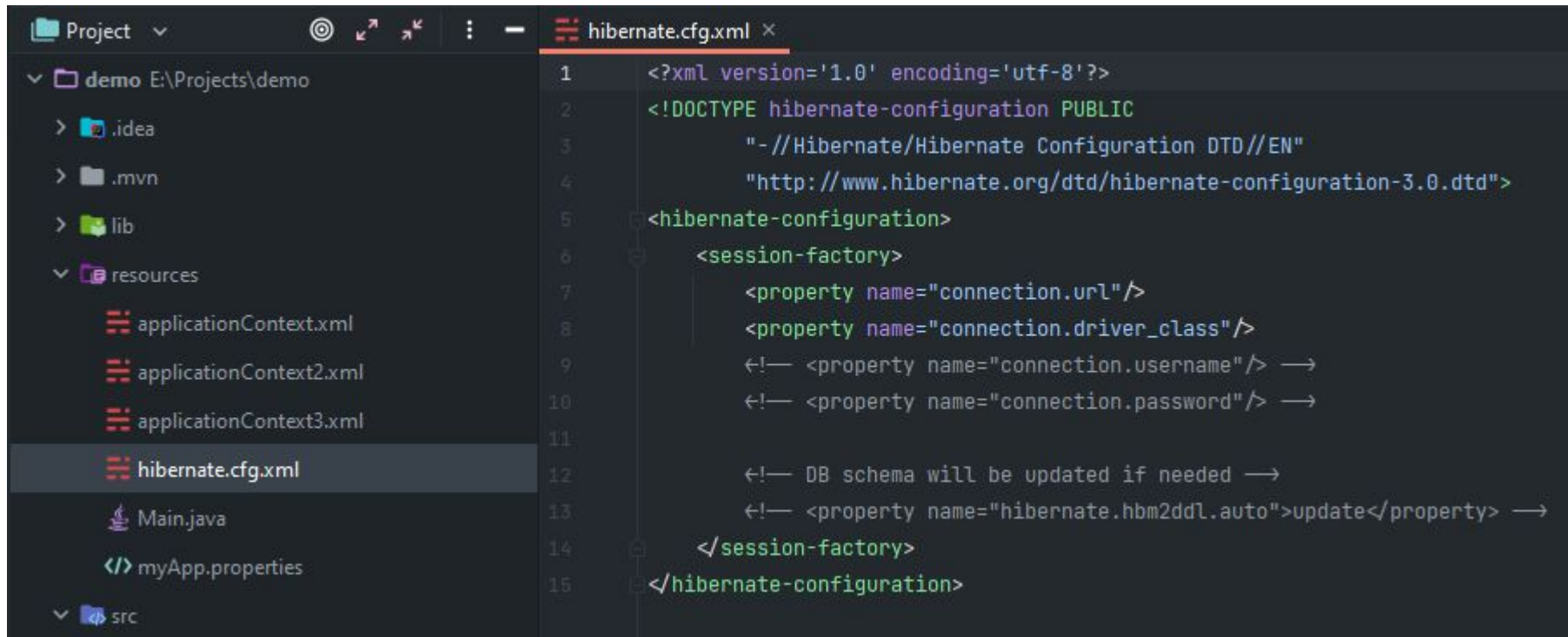
Подключение Hibernate

В папке lib должен появиться jar-файл Hibernate



Подключение Hibernate

Также будет создан файл конфигурации Hibernate



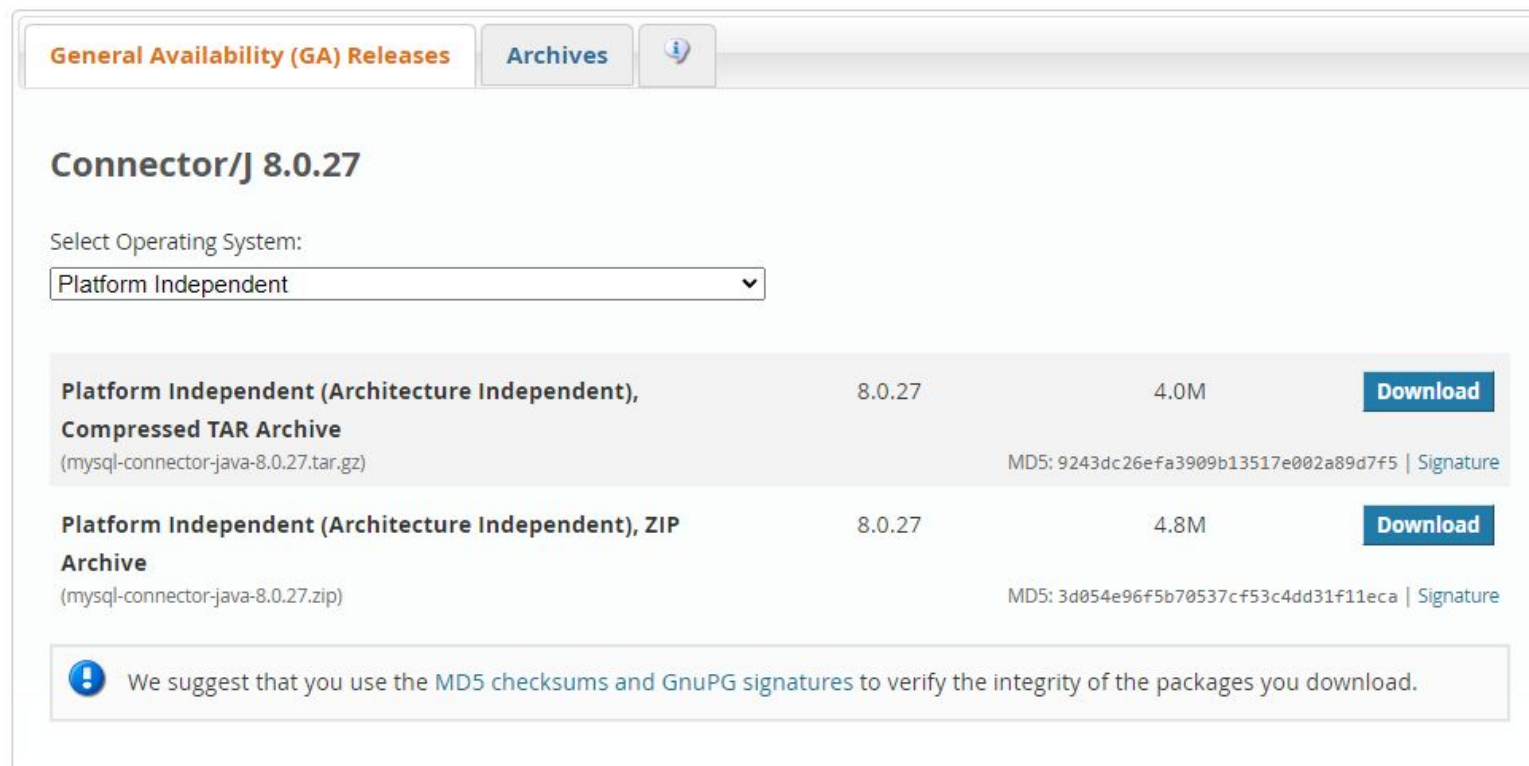
The screenshot shows an IDE window with a project explorer on the left and a code editor on the right. The project explorer shows a project named 'demo' with a 'resources' folder containing 'hibernate.cfg.xml'. The code editor displays the XML configuration for Hibernate, including the DOCTYPE declaration, the hibernate-configuration root element, and the session-factory configuration with properties for connection.url, connection.driver_class, connection.username, connection.password, and hibernate.hbm2ddl.auto.

```
1 <?xml version='1.0' encoding='utf-8'?>
2 <!DOCTYPE hibernate-configuration PUBLIC
3     "-//Hibernate/Hibernate Configuration DTD//EN"
4     "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
5 <hibernate-configuration>
6     <session-factory>
7         <property name="connection.url"/>
8         <property name="connection.driver_class"/>
9         <!-- <property name="connection.username"/> -->
10        <!-- <property name="connection.password"/> -->
11
12        <!-- DB schema will be updated if needed -->
13        <!-- <property name="hibernate.hbm2ddl.auto">update</property> -->
14    </session-factory>
15 </hibernate-configuration>
```



Подключение Hibernate

Также необходимо подключить MySQL JDBC Driver. Его можно скачать на сайте <https://dev.mysql.com/downloads/connector/j/>



General Availability (GA) Releases Archives ⓘ

Connector/J 8.0.27

Select Operating System:
Platform Independent ▾

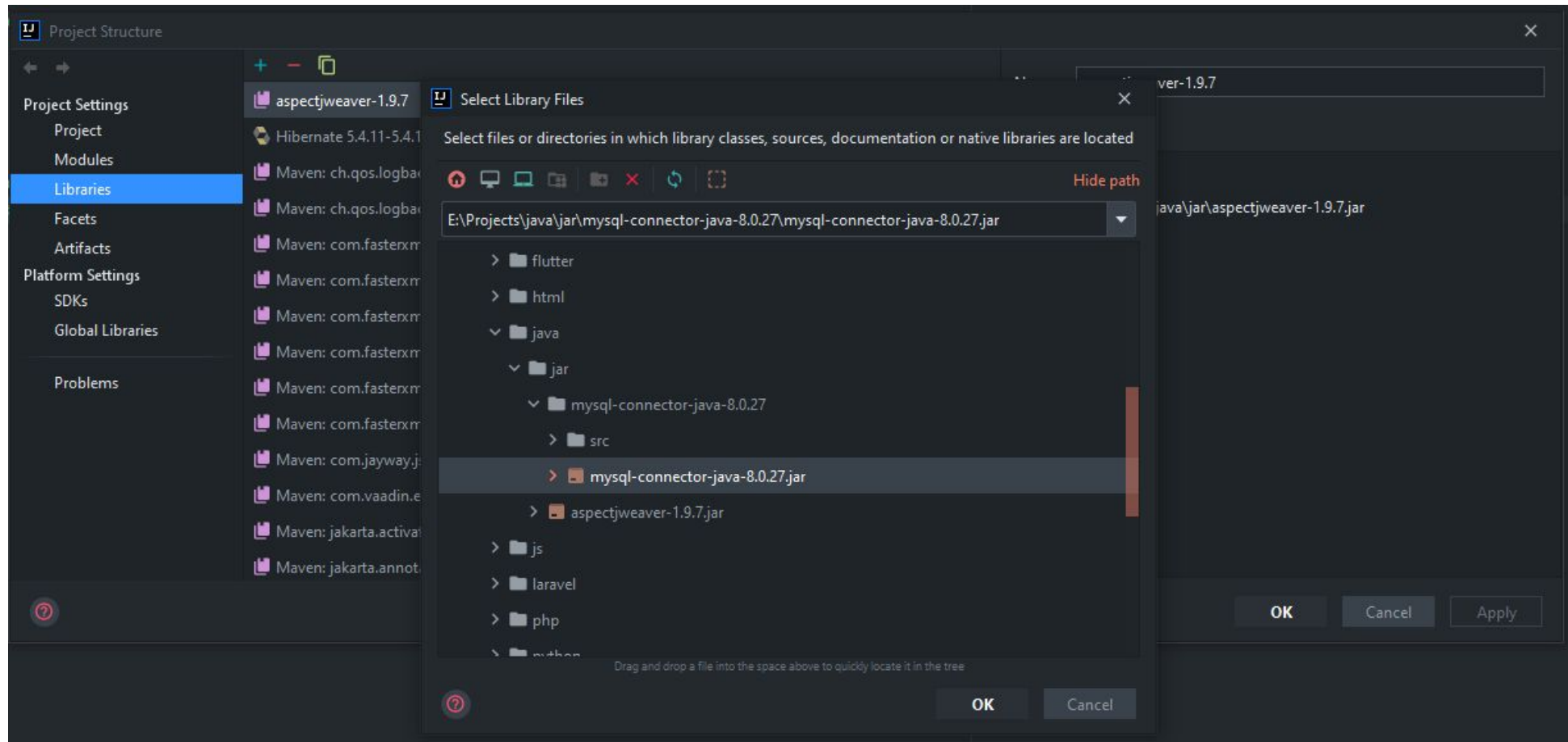
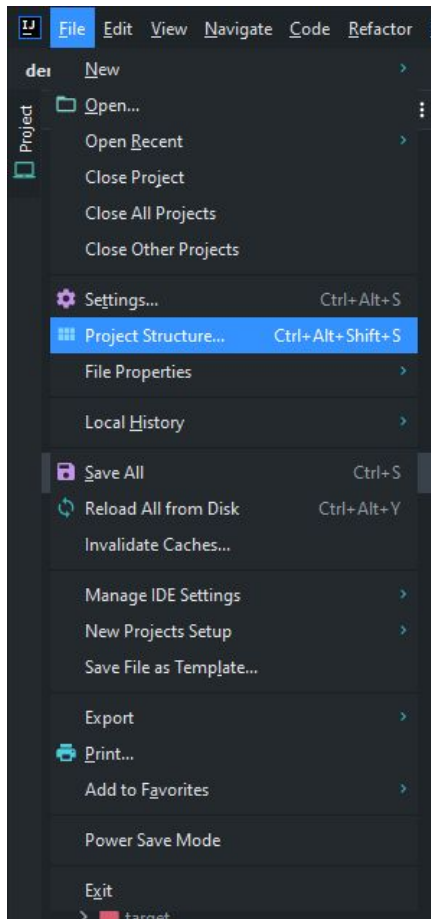
Platform Independent (Architecture Independent), Compressed TAR Archive <small>(mysql-connector-java-8.0.27.tar.gz)</small>	8.0.27	4.0M	Download
<small>MD5: 9243dc26efa3909b13517e002a89d7f5 Signature</small>			
Platform Independent (Architecture Independent), ZIP Archive <small>(mysql-connector-java-8.0.27.zip)</small>	8.0.27	4.8M	Download
<small>MD5: 3d054e96f5b70537cf53c4dd31f11eca Signature</small>			

! We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download.



Подключение Hibernate

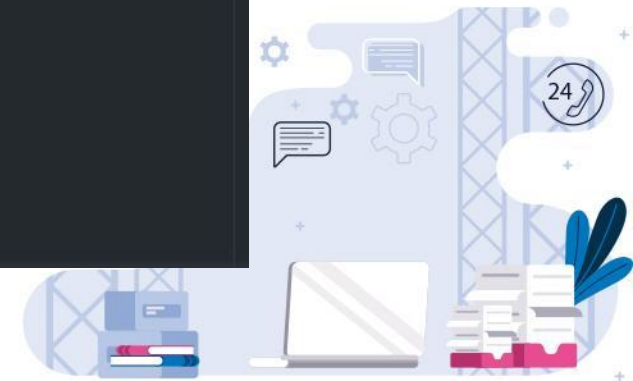
Теперь необходимо добавить его в проект.



Подключение Hibernate

После этого необходимо добавить конфигурацию в файл hibernate.cfg.xml

```
hibernate.cfg.xml ×
1  <?xml version='1.0' encoding='utf-8'?>
2  <!DOCTYPE hibernate-configuration PUBLIC
3      "-//Hibernate/Hibernate Configuration DTD//EN"
4      "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
5  <hibernate-configuration>
6      <session-factory>
7          <property name="connection.url">jdbc:mysql://localhost:3306/my_db?useSSL=false&serverTimezone=UTC</property>
8          <property name="connection.driver_class">com.mysql.cj.jdbc.Driver</property>
9          <property name="connection.username">root</property>
10         <property name="connection.password">root</property>
11
12         <property name="current_session_context_class">thread</property>
13         <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
14         <property name="show_sql">>true</property>
15     </session-factory>
16 </hibernate-configuration>
```



Создание связи между классом и таблицей

Конфигурировать связь между классом и таблицей можно двумя способами:

1. С помощью XML-файла (старый способ)
2. С помощью Java-аннотаций



Создание связи между классом и таблицей

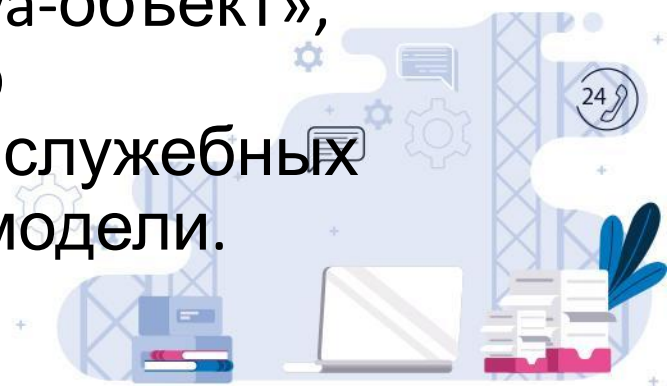
Hibernate использует концепцию Entity класс.

Entity класс – это Java класс, который отображает информацию определенной таблицы в БД.

Другое определение:

Entity класс – это POJO класс, в котором мы используем определенные Hibernate аннотации для связи класса с таблицей в БД.

POJO (англ. Plain Old Java Object) — «старый добрый Java-объект», простой Java-объект, не унаследованный от какого-то специфического объекта и не реализующий никаких служебных интерфейсов сверх тех, которые нужны для бизнес-модели.



Создание связи между классом и таблицей

Создадим класс Employee

```
1 package com.donnu.demo.hibernate_test.entity;
2
3 import javax.persistence.*;
4
5 @Entity
6 @Table(name="employees")
7 public class Employee {
8     @Id
9     @Column(name="id")
10    private int id;
11    @Column(name="name")
12    private String name;
13    @Column(name="surname")
14    private String surname;
15    @Column(name="department")
16    private String department;
17    @Column(name="salary")
18    private int salary;
19
20    public Employee() {
21    }
22
23    public Employee(String name, String surname, String department, int salary) {
24        this.name = name;
25        this.surname = surname;
26        this.department = department;
27        this.salary = salary;
28    }
29
```

```
30
31    public int getId() { return id; }
32
33
34    public void setId(int id) { this.id = id; }
35
36
37
38
39    public String getName() { return name; }
40
41
42    public void setName(String name) { this.name = name; }
43
44
45
46    public String getSurname() { return surname; }
47
48
49
50    public void setSurname(String surname) { this.surname = surname; }
51
52
53
54
55    public String getDepartment() { return department; }
56
57
58    public void setDepartment(String department) { this.department = department; }
59
60
61
62    public int getSalary() { return salary; }
63
64
65
66    public void setSalary(int salary) { this.salary = salary; }
67
68
69
70
71    @Override
72    public String toString() {
73        return "Employee{" +
74            "id=" + id +
75            ", name='" + name + '\'' +
76            ", surname='" + surname + '\'' +
77            ", department='" + department + '\'' +
78            ", salary=" + salary +
79            '}';
80    }
81
```



Создание связи между классом и таблицей

Аннотация **@Entity** говорит о том, что данный класс будет иметь отображение в БД.

Аннотация **@Table** говорит о том, к какой именно таблице мы привязываем класс.

Аннотация **@Column** говорит о том, к какому именно столбцу из таблицы мы привязываем поле класса.

Аннотация **@Id** говорит о том, что столбец связанный с данным полем является первичным ключом.



Создание связи между классом и таблицей

JPA (Java Persistence API) – это стандартная спецификация, которая описывает систему для управления сохранением Java объектов в таблицы БД.

Hibernate – самая популярная реализация спецификации JPA.



Сохранение Java объектов в БД

Создадим класс Test1 и создадим в нем SessionFactory

```
package com.donnu.demo.hibernate_test;

import com.donnu.demo.hibernate_test.entity.Employee;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class Test1 {
    public static void main(String[] args) {
        SessionFactory factory = new Configuration()
            .configure("hibernate.cfg.xml")
            .addAnnotatedClass(Employee.class)
            .buildSessionFactory();
    }
}
```



Сохранение Java объектов в БД

SessionFactory – это фабрика по производству сессий.

SessionFactory читает **hibernate.cfg.xml** после чего SessionFactory узнает как должны создаваться сессии.

В Java приложении достаточно создать SessionFactory один раз, а потом можно его переиспользовать.



Сохранение Java объектов в БД

Теперь необходимо создать сессию

```
public class Test1 {  
    public static void main(String[] args) {  
        SessionFactory factory = new Configuration()  
            .configure("hibernate.cfg.xml")  
            .addAnnotatedClass(Employee.class)  
            .buildSessionFactory();  
  
        Session session = factory.getCurrentSession();  
    }  
}
```



Сохранение Java объектов в БД

Session – это обертка вокруг подключения к базе с помощью JDBC.

Она может быть получена с помощью `SessionFactory`.

`Session` – это основа для работы с БД. Именно с ее помощью будут выполняться операции добавления, удаления и т.д.

Жизненный цикл сессии, как правило, не велик. Мы получаем сессию, делаем что-то с ее помощью, после чего сессия становится ненужной.



Сохранение Java объектов в БД

Сохраним сотрудника в базу. Стоит помнить, что при работе могут быть выброшены исключения, а фабрику нужно закрыть в любом случае. Поместим закрытие фабрики в блок `finally`.

```
public class Test1 {  
    public static void main(String[] args) {  
        SessionFactory factory = new Configuration()  
            .configure("hibernate.cfg.xml")  
            .addAnnotatedClass(Employee.class)  
            .buildSessionFactory();  
  
        try {  
            Session session = factory.getCurrentSession();  
  
            Employee employee = new Employee(name: "Ivan", surname: "Ivanov", department: "Design", salary: 80000);  
  
            session.beginTransaction();  
            session.save(employee);  
            session.getTransaction().commit();  
        }  
        finally {  
            factory.close();  
        }  
    }  
}
```



Сохранение Java объектов в БД

После запуска видим:

```
01:27:09.248 [main] DEBUG org.hibernate.internal.util.EntityPrinter - Listing entities:  
01:27:09.248 [main] DEBUG org.hibernate.internal.util.EntityPrinter - com.donnu.demo.hibernate_test.entity.Employee{surname=Ivanov, name=Ivan, id=0, department=Design, salary=80000}  
01:27:09.254 [main] DEBUG org.hibernate.SQL - insert into employees (department, name, salary, surname, id) values (?, ?, ?, ?, ?)  
Hibernate: insert into employees (department, name, salary, surname, id) values (?, ?, ?, ?, ?)
```

✓ Отображение строк 0 - 0 (1 всего, Запрос занял 0,0005 сек.)

```
SELECT * FROM `employees`
```

Показать все | Количество строк: 50 | Фильтровать строки: Поиск в таблице

+ Параметры

	id	name	surname	department	salary
<input type="checkbox"/>	1	Ivan	Ivanov	Design	80000

















Отметить все | С отмеченными:

Показать все | Количество строк: 50 | Фильтровать строки: Поиск в таблице



Сохранение Java объектов в БД

Структура таблицы:

	#	Имя	Тип	Сравнение	Атрибуты	Null	По умолчанию	Комментарии	Дополнительно	Действие
<input type="checkbox"/>	1	id 	int			Нет	Нет		AUTO_INCREMENT	  
<input type="checkbox"/>	2	name	varchar(255)	utf8mb4_0900_ai_ci		Нет	Нет			  
<input type="checkbox"/>	3	surname	varchar(255)	utf8mb4_0900_ai_ci		Нет	Нет			  
<input type="checkbox"/>	4	department	varchar(255)	utf8mb4_0900_ai_ci		Нет	Нет			  
<input type="checkbox"/>	5	salary	int			Нет	Нет			  



Генерация значений для столбца Primary Key

Столбец **Primary Key** содержит уникальное значение и не может быть null.

Аннотация **@GeneratedValue** описывает стратегию генерации значений для столбца Primary Key.



Генерация значений для столбца Primary Key

GenerationType.IDENTITY полагается на автоматическое увеличение столбца по правилам, прописанным в БД. Хороший вариант с точки зрения производительности.

```
@Id
@GeneratedValue(strategy=GenerationType.IDENTITY)
@Column(name="id")
private int id;
```

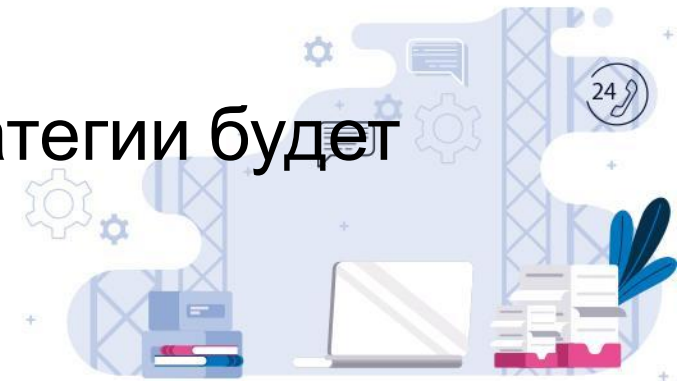


Генерация значений для столбца Primary Key

GenerationType.SEQUENCE полагается на работу sequence, созданного в БД. Тоже очень эффективная с точки зрения производительности стратегия. В MySQL sequence не поддерживается, но если вы работаете с Oracle, то можете использовать этот способ.

GenerationType.TABLE полагается на значение столбца таблицы БД. Цель такой таблицы – поддержка уникальности значений (при получении значение меняется, например, увеличивается на 1). Устаревший способ, не эффективен по производительности.

GenerationType.AUTO дефолтный тип. Выбор стратегии будет зависеть от типа базы, с которой мы работаем.



Получение Java объектов из БД

Прежде всего стоит отметить, что мы можем получить id сохраненного объекта непосредственно после его сохранения

```
session.beginTransaction();  
session.save(employee);  
session.getTransaction().commit();  
  
System.out.println(employee);
```

```
Employee{id=3, name='Ivan', surname='Ivanov', department='Design', salary=80000}
```



Получение Java объектов из БД

Создадим класс Test2

Вывод:

```
Employee{id=4, name='Elena', surname='Petrova', department='IT', salary=100000}
```

```
public class Test2 {  
    public static void main(String[] args) {  
        SessionFactory factory = new Configuration()  
            .configure("hibernate.cfg.xml")  
            .addAnnotatedClass(Employee.class)  
            .buildSessionFactory();  
  
        try {  
            Session session = factory.getCurrentSession();  
  
            Employee employee = new Employee(name: "Elena", surname: "Petrova", department: "IT", salary: 100000);  
  
            session.beginTransaction();  
            session.save(employee);  
            session.getTransaction().commit();  
  
            int employeeId = employee.getId();  
  
            // получаем новую сессию  
            session = factory.getCurrentSession();  
            session.beginTransaction();  
            Employee employeeFromDb = session.get(Employee.class, employeeId);  
            session.getTransaction().commit();  
  
            System.out.println(employeeFromDb);  
        }  
        finally {  
            factory.close();  
        }  
    }  
}
```

Получение Java объектов из БД

Рассмотрим ситуацию, когда работника с таким id нет

```
try {  
    Session session = factory.getCurrentSession();  
    session.beginTransaction();  
    Employee employeeFromDb = session.get(Employee.class, serializable: 10);  
    session.getTransaction().commit();  
  
    System.out.println(employeeFromDb);  
}  
finally {  
    factory.close();  
}
```

Вывод:

```
null
```



Получение Java объектов из БД

Можно выполнить несколько действий в рамках одной сессии

```
try {  
    Session session = factory.getCurrentSession();  
  
    Employee employee = new Employee( name: "Oleg", surname: "Veshiy", department: "Kniaz", salary: 912);  
  
    session.beginTransaction();  
    session.save(employee);  
    //session.getTransaction().commit();  
  
    int employeeId = employee.getId();  
  
    // получаем новую сессию  
    //session = factory.getCurrentSession();  
    //session.beginTransaction();  
    Employee employeeFromDb = session.get(Employee.class, employeeId);  
    session.getTransaction().commit();  
  
    System.out.println(employeeFromDb);  
}  
finally {  
    factory.close();  
}
```

```
Employee{id=5, name='Oleg', surname='Veshiy', department='Kniaz', salary=912}
```



Получение Java объектов из БД

Рассмотрим получение не по id. Для этого используется HQL.

HQL (Hibernate Query Language) очень похож на SQL.

```
SessionFactory factory = new Configuration()
    .configure("hibernate.cfg.xml")
    .addAnnotatedClass(Employee.class)
    .buildSessionFactory();

try {
    Session session = factory.getCurrentSession();
    session.beginTransaction();

    // Обратите внимание, что Employee не имя таблицы, а имя класса
    List<Employee> employees = session.createQuery( s: "from Employee")
        .getResultList();

    for (Employee e: employees)
        System.out.println(e);

    session.getTransaction().commit();
}
finally {
    factory.close();
}
```

```
Employee{id=1, name='Ivan', surname='Ivanov', department='Design', salary=80000}
Employee{id=2, name='Ivan', surname='Ivanov', department='Design', salary=80000}
Employee{id=3, name='Ivan', surname='Ivanov', department='Design', salary=80000}
Employee{id=4, name='Elena', surname='Petrova', department='IT', salary=100000}
Employee{id=5, name='Oleg', surname='Veshiy', department='Kniaz', salary=912}
```



Получение Java объектов из БД

В запрос можно добавить условие

```
// Обратите внимание, что Employee не имя таблицы, а имя класса
List<Employee> employees = session.createQuery( s: "from Employee where name = 'Ivan'")
    .getResultList();
```

```
Employee{id=1, name='Ivan', surname='Ivanov', department='Design', salary=80000}
Employee{id=2, name='Ivan', surname='Ivanov', department='Design', salary=80000}
Employee{id=3, name='Ivan', surname='Ivanov', department='Design', salary=80000}
```

And:

```
// Обратите внимание, что Employee не имя таблицы, а имя класса
List<Employee> employees = session.createQuery( s: "from Employee where name = 'Ivan' AND salary > 75000")
    .getResultList();
```



Изменение Java объектов в БД

Для изменения поля в записи базы данных мы можем использовать setter

```
Session session = factory.getCurrentSession();
session.beginTransaction();

Employee employee = session.get(Employee.class, serializable: 1);
employee.setSalary(50000);

session.getTransaction().commit();
```

				id	name	surname	department	salary
<input type="checkbox"/>				1	Ivan	Ivanov	Design	50000
<input type="checkbox"/>				2	Ivan	Ivanov	Design	80000
<input type="checkbox"/>				3	Ivan	Ivanov	Design	70000
<input type="checkbox"/>				4	Elena	Petrova	IT	100000
<input type="checkbox"/>				5	Oleg	Veshiy	Kniaz	912



Изменение Java объектов в БД

Обновление нескольких записей

```
SessionFactory factory = new Configuration()
    .configure("hibernate.cfg.xml")
    .addAnnotatedClass(Employee.class)
    .buildSessionFactory();
try {
    Session session = factory.getCurrentSession();
    session.beginTransaction();

    session.createQuery("update Employee set salary=80000 where name = 'Ivan'").executeUpdate();

    session.getTransaction().commit();
}
finally {
    factory.close();
}
```

				id	name	surname	department	salary
<input type="checkbox"/>				1	Ivan	Ivanov	Design	80000
<input type="checkbox"/>				2	Ivan	Ivanov	Design	80000
<input type="checkbox"/>				3	Ivan	Ivanov	Design	80000
<input type="checkbox"/>				4	Elena	Petrova	IT	100000
<input type="checkbox"/>				5	Oleg	Veshiy	Kniaz	912



Удаление Java объектов из БД

Удаление работника найденного по id. Если работник не найден будет ошибка.

```
Session session = factory.getCurrentSession();
session.beginTransaction();

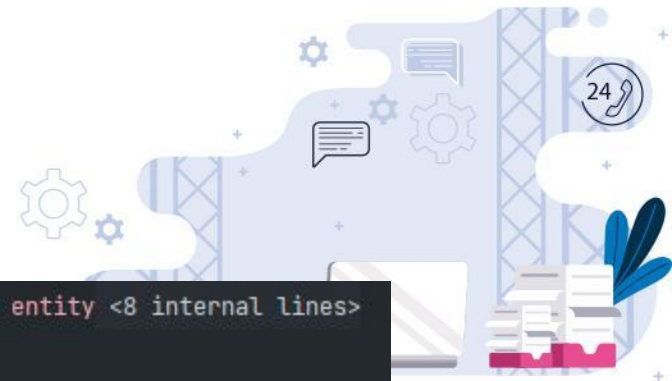
Employee employee = session.get(Employee.class, serializable: 1);

session.delete(employee);

session.getTransaction().commit();
```

				id	name	surname	department	salary
<input type="checkbox"/>				2	Ivan	Ivanov	Design	80000
<input type="checkbox"/>				3	Ivan	Ivanov	Design	80000
<input type="checkbox"/>				4	Elena	Petrova	IT	100000
<input type="checkbox"/>				5	Oleg	Veshiy	Kniaz	912









```
Exception in thread "main" java.lang.IllegalArgumentException: attempt to create delete event with null entity <8 internal lines>
at com.donnu.demo.hibernate_test.Test2.main(Test2.java:22)
```



Удаление Java объектов из БД

Удаление нескольких работников

```
Session session = factory.getCurrentSession();  
session.beginTransaction();  
  
session.createQuery("delete Employee where name = 'Ivan'").executeUpdate();  
  
session.getTransaction().commit();
```

				id	name	surname	department	salary	
<input type="checkbox"/>					4	Elena	Petrova	IT	100000
<input type="checkbox"/>					5	Oleg	Veshiy	Kniaz	912

