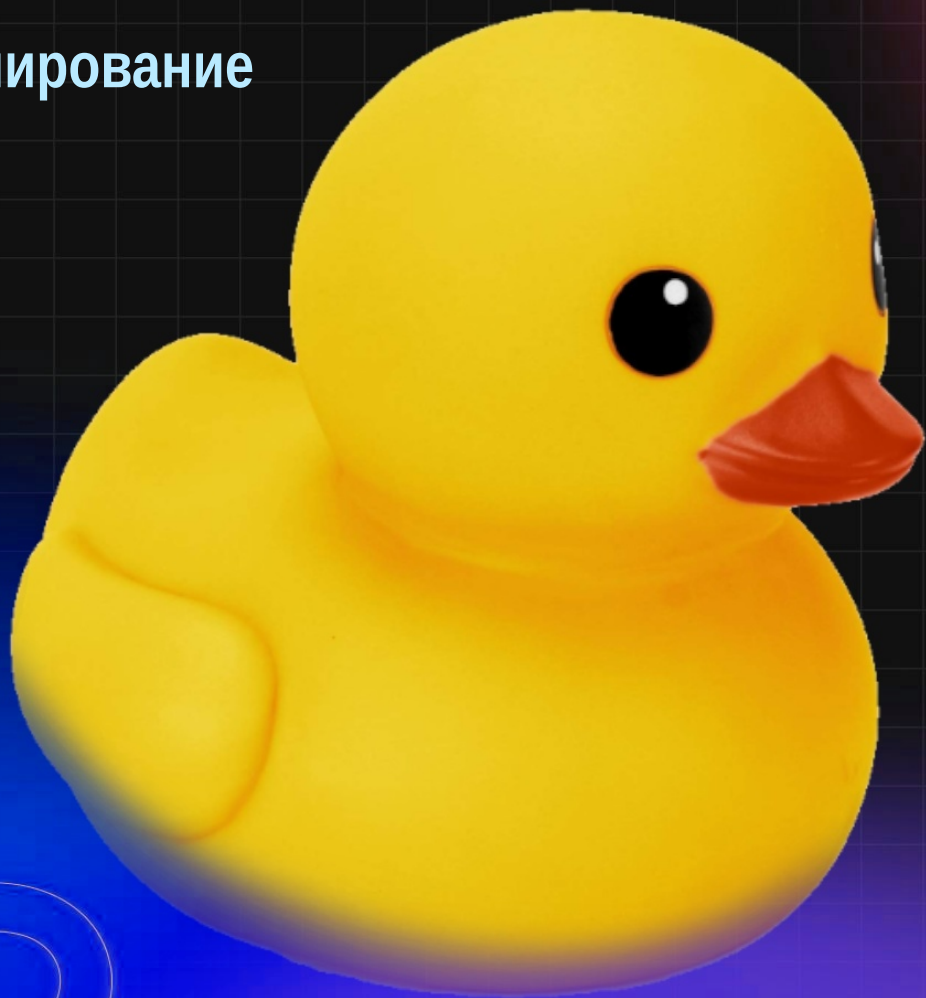


Программирование  
2 семестр  
2024

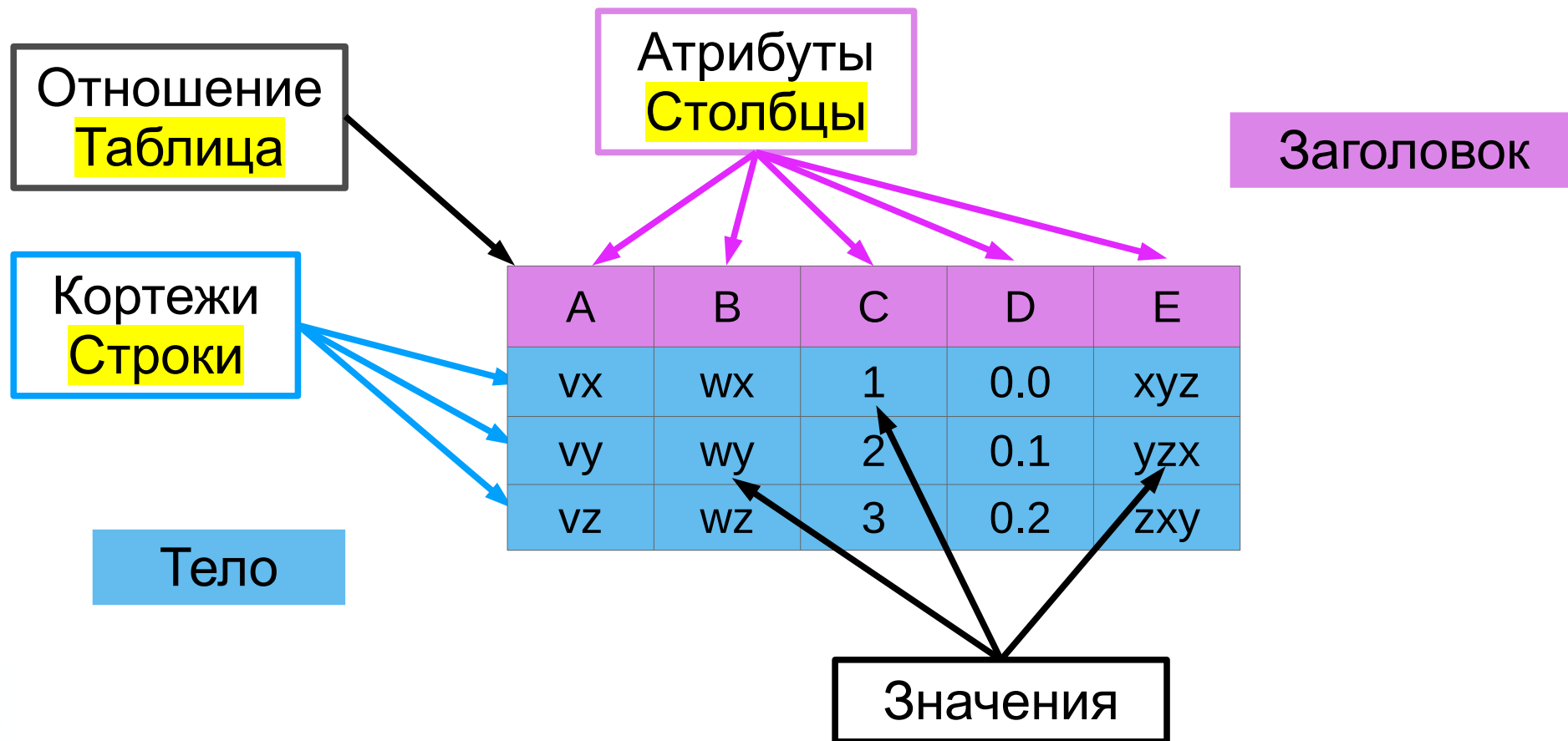


ІТМО

Взаимодействие  
с базами данных

- БД — структурно организованные данные о предметной области, хранящиеся вместе с информацией о данных и их взаимосвязях.
- СУБД — вычислительная система для создания и использования баз данных.
- Реляционная БД — база данных, основанная на реляционной модели данных.

- Отношение (relation) — структура данных, состоящая из заголовка и тела.
  - ❖ Заголовок отношения — множество атрибутов
  - ❖ Тело отношения — множество кортежей, содержащих значения атрибутов
- Свойства отношения
  - ❖ Каждый атрибут имеет тип, значения соответствуют типу
  - ❖ Атрибуты не повторяются, и их порядок не имеет значения
  - ❖ Кортежи не повторяются, и их порядок не имеет значения



# Пример базы данных

students

student_id	name	group
289001	Иванов Петр	P3110
289002	Петров Сидор	P3130
289999	Сидоров Иван	R3140

groups

group	faculty
P3110	ПИиКТ
P3130	ПИиКТ
R3140	СУиР

grades

student_id	course_id	grade
289001	1	A
289001	2	B
289002	1	E
289002	2	A
289999	1	C
288999	3	D

courses

course_id	name	semester	type
1	Программирование	1	З
2	ОПД	2	Э
3	Физика	2	Э

- SQL (Structured Query Language) — декларативный язык для описания, изменения и получения данных из реляционных баз данных.

- DDL — Data definition language

```
CREATE TABLE weather (  
    city        VARCHAR(80),  
    temp_lo     INT,          -- low temperature  
    temp_hi     INT,          -- high temperature  
    prcp        REAL,         -- precipitation  
    date        DATE  
);  
  
CREATE TABLE IF NOT EXISTS persons  
  
DROP TABLE weather;
```

- Ограничения
  - ❖ Типы данных
    - INT, SMALLINT, REAL, DOUBLE, CHAR(n), VARCHAR(n), DATE, TIME, ...
  - ❖ Возможные значения в столбце
    - NOT NULL, UNIQUE, CHECK (age >= 18)
  - ❖ Ключи
    - Первичный ключ
      - ▶ PRIMARY KEY (UNIQUE, NOT NULL)
    - Внешние ключи
      - ▶ FOREIGN KEY (REFERENCES)



- DML — Data manipulation language
  - ❖ INSERT
  - ❖ UPDATE
  - ❖ DELETE

- Вставка данных в таблицу

```
INSERT INTO weather VALUES ('Oslo', 46, 50, 0.25, '2021-11-27');
```

```
INSERT INTO weather (date, city, temp_hi, temp_lo)  
VALUES ('2021-11-29', 'Helsinki', 54, 37);
```

```
COPY persons TO file;
```

```
COPY persons FROM file;
```

- Обновление данных в таблице

```
UPDATE weather  
  SET temp_hi = temp_hi - 2,  temp_lo = temp_lo - 2  
 WHERE date > '2021-11-28';
```

- Удаление данных из таблицы

```
DELETE FROM weather WHERE city = 'Oslo';
```

```
DELETE FROM persons;
```

- Выборка данных - запрос

```
SELECT * FROM students;  
SELECT name, group FROM students;  
SELECT * FROM students WHERE group = 'P3110';  
SELECT * FROM students ORDER BY name;  
SELECT DISTINCT type FROM courses;  
SELECT COUNT(*) FROM students;
```

- Соединение таблиц

```
SELECT name, faculty FROM students JOIN groups
      ON students.group = groups.group;  -- USING(group);
```

students

student_id	name	group
289001	Иванов Петр	P3110
289002	Петров Сидор	P3130
289999	Сидоров Иван	R3140

groups

group	faculty
P3110	ПИиКТ
P3130	ПИиКТ
R3140	СУиР

JOIN

student_id	name	group	faculty
289001	Иванов Петр	P3110	ПИиКТ
289002	Петров Сидор	P3130	ПИиКТ
289999	Сидоров Иван	R3140	СУиР

- helios

- ❖ `psql -h pg studs`

- ❖ пароль в файле `.pgpass`

- `host:port:dbname:login:PaSsw0rD`

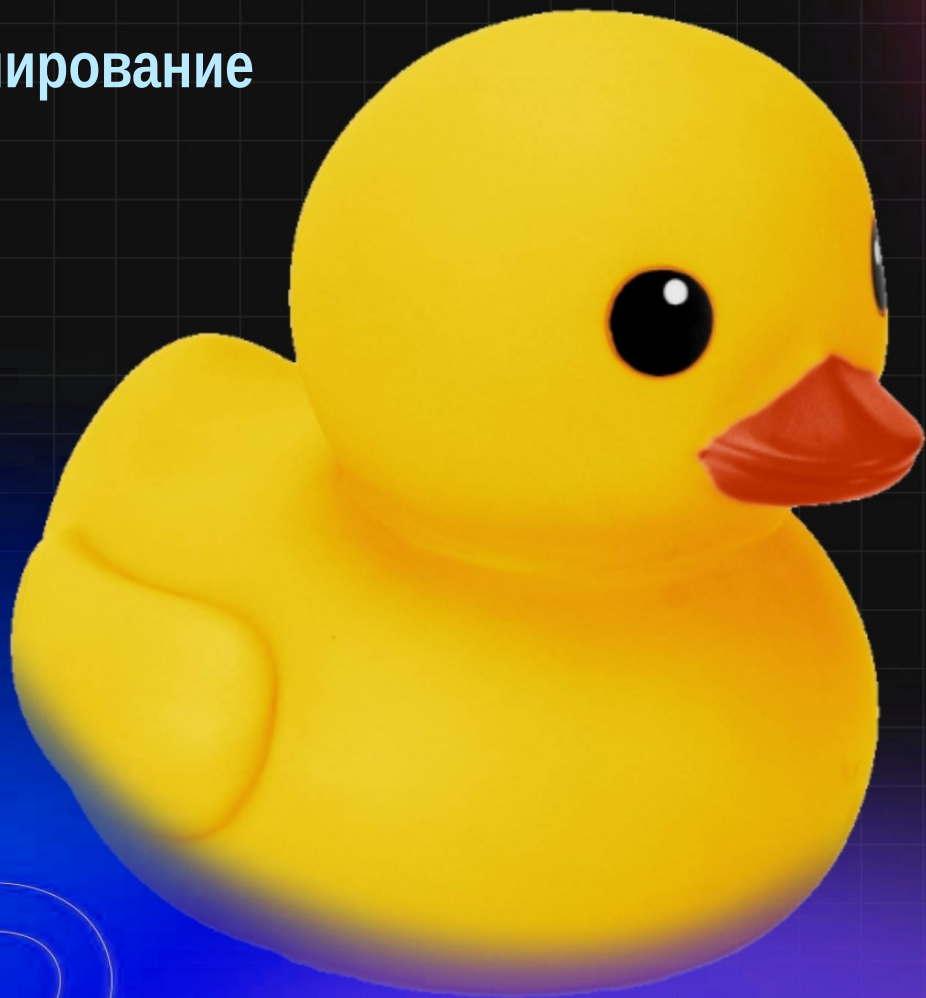
- ❖ драйвер - `/usr/local/share/java/classes/postgresql.jar`

- <https://postgresql.org>

- ❖ `\?` - помощь по командам `psql`

- ❖ `\h` - помощь по командам SQL

Программирование  
2 семестр  
2024



**ІТМО**

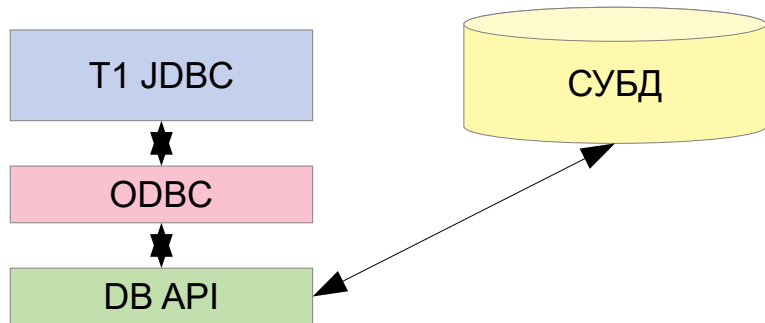
**ОСНОВЫ JDBC**



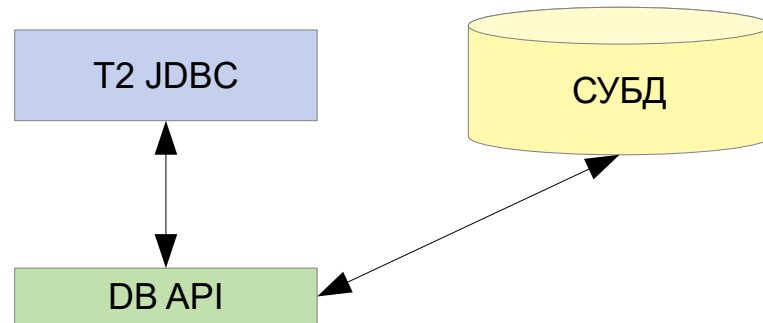
- Много разных СУБД
  - ❖ API для каждой базы отдельно
  - ❖ единый интерфейс работы с базами + драйвер для конкретной базы
- Реализации
  - ❖ ODBC — Open Database Connectivity
  - ❖ JDBC — Java Database Connectivity

- JDBC — Java DataBase Connectivity
- JDBC API — высокоуровневый интерфейс для доступа к данным
- JDBC Driver API — низкоуровневый интерфейс для создания драйверов
- Пакеты `java.sql` (Core) и `javax.sql` (Extension)
- Стандарт взаимодействия с СУБД

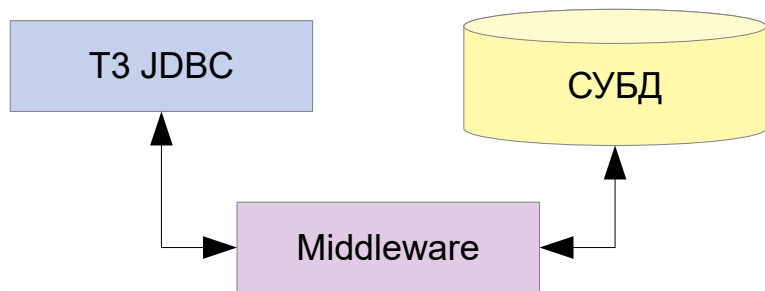
- Тип 1 — мост ODBC



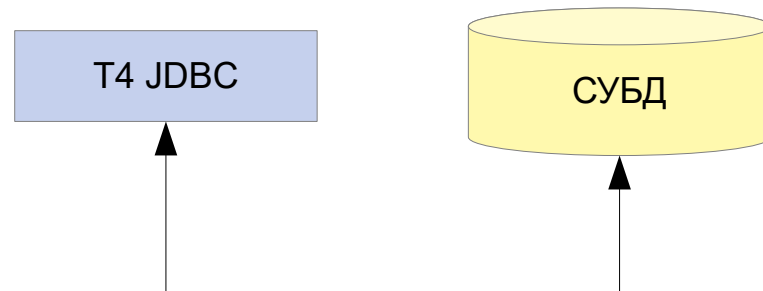
- Тип 2 — DB API



- Тип 3 — middleware



- Тип 4 — pure Java



```
Connection connection = DriverManager.getConnection(...);

Statement statement = connection.createStatement();

ResultSet resultSet = statement.executeQuery("SELECT ...");

while (resultSet.next()) {
    // получение и обработка данных
}

resultSet.close();
statement.close();
connection.close();
```

- `java.sql.Driver`
  - ❖ Отвечает за связь с БД
  - ❖ Метод `Connection connect(String url, Properties info)`
  - ❖ Используется для написания драйверов для СУБД

- Управляет списком драйверов
- Загрузка драйвера
  - ❖ `Class.forName()`
  - ❖ `jdbc.drivers=`
- Неявная загрузка с помощью `ServiceLoader`
  - ❖ `META-INF/services/java.sql.Driver`

- Метод `Connection getConnection(String url, ...)`
  - ❖ `url = jdbc:protocol://host:port/database`
    - `jdbc:postgresql://db:5432/studs`
  - ❖ `getConnection(String url, Properties info)`
    - `Properties info = new Properties();`
    - `info.load(new FileInputStream("db.cfg"));`  
файл db.cfg

```
user = s999999
password = sss999
```
  - ❖ `getConnection(String url, String name, String pass)`

- Абстракция соединения (сессия)

- ❖ методы:

- `Statement` `createStatement()`
- `PreparedStatement` `prepareStatement(String sql)`
- `CallableStatement` `prepareCall(String sql)`
- `DatabaseMetaData` `getMetaData()`



- Statement

- ❖ Статический SQL-запрос
- ❖ Запрос передается как параметр при выполнении

```
String query = "SELECT * FROM table WHERE id = 15";
```

```
Statement st = connection.createStatement();
```

```
st.executeQuery(query);
```

- **PreparedStatement** extends Statement

- ❖ **Динамический запрос** с подстановкой

- ❖ Запрос передается как параметр при создании

```
String query = "SELECT * FROM table WHERE id = ?";
```

```
PreparedStatement ps = connection.prepareStatement(query);
```

```
ps.setInt(1, 15); // 1 — номер параметра, 15 — значение
```

```
SELECT * FROM table WHERE id = 15
```

- Предотвращает SQL-инъекции

- CallableStatement extends PreparedStatement

- ❖ Вызов хранимой процедуры

SQL: CREATE PROCEDURE

cs = prepareCall("CALL getResult (?)");

cs.setInt(1, 15);

cs.registerOutParameter(1, Types.INTEGER);

...

int result = cs.getInt(1);

CALL getResult(15);

- `ResultSet executeQuery(String sql) // Statement`
- `ResultSet executeQuery() // PreparedStatement (Callable)`
  - ❖ для выполнения SELECT
  - ❖ возвращает ResultSet

```
ResultSet resultSet = statement.executeQuery();
```

- `int executeUpdate(String sql) // Statement`
- `int executeUpdate() // PreparedStatement (Callable)`
  - ❖ для выполнения INSERT, UPDATE, DELETE + DDL
  - ❖ возвращает количество измененных строк
  - ❖ Для запросов DDL возвращает 0

```
int updateCount = statement.executeQuery();
```

- `boolean execute(String sql) // Statement`
- `boolean execute() // PreparedStatement (Callable)`
  - ❖ для выполнения любых запросов

```
if (statement.execute()) {  
    ResultSet resultSet = statement.getResultSet();  
} else {  
    int updateCount = statement.getUpdateCount();  
}
```



- Connection
  - ❖ `setAutoCommit(true/false)`
  - ❖ `commit()`
  - ❖ `rollback()`
  - ❖ `setSavepoint()`

- Connection

- ❖ `setAutoCommit(true/false)`
- ❖ `commit()`
- ❖ `rollback()`
- ❖ `setSavepoint()`

- Statement

- ❖ `addBatch(String sql)`
- ❖ `clearBatch()`
- ❖ `executeBatch()`



- Получение данных из **ResultSet**

```
ResultSet rs = preparedStatement.executeQuery();
```

```
while (rs.next()) {  
    String name = rs.getString(1); // по номеру столбца  
    int id = rs.getInt("id");      // по имени столбца  
}
```

- **ResultSet**

- ❖ Connection.**createStatement**(sql, type, concurrency, holdability)
- ❖ ResultSetType
  - **TYPE\_FORWARD\_ONLY**
  - **TYPE\_SCROLL\_INSENSITIVE**
  - **TYPE\_SCROLL\_SENSITIVE**
- ❖ ResultSetConcurrency,
  - **CONCUR\_READ\_ONLY**
  - **CONCUR\_UPDATABLE**
- ❖ ResultSetHoldability
  - **HOLD\_CURSORS\_OVER\_COMMIT**
  - **CLOSE\_CURSORS\_AT\_COMMIT**

- Навигация

- ❖ next()
- ❖ previous()
- ❖ first()
- ❖ last()
- ❖ beforeFirst()
- ❖ afterLast()
- ❖ relative(int row)
- ❖ absolute(int row)
- ❖ moveToInsertRow()

- Получение данных

- ❖ getString(int)
- ❖ getString(String)
- ❖ getInt(int)
- ❖ getInt(String)
- ❖ getBoolean
- ❖ getLong
- ❖ getDouble
- ❖ getArray (SQL Array)
- ❖ getDate
- ❖ getTimestamp
- ❖ getReader
- ❖ ...

- Обновление строк

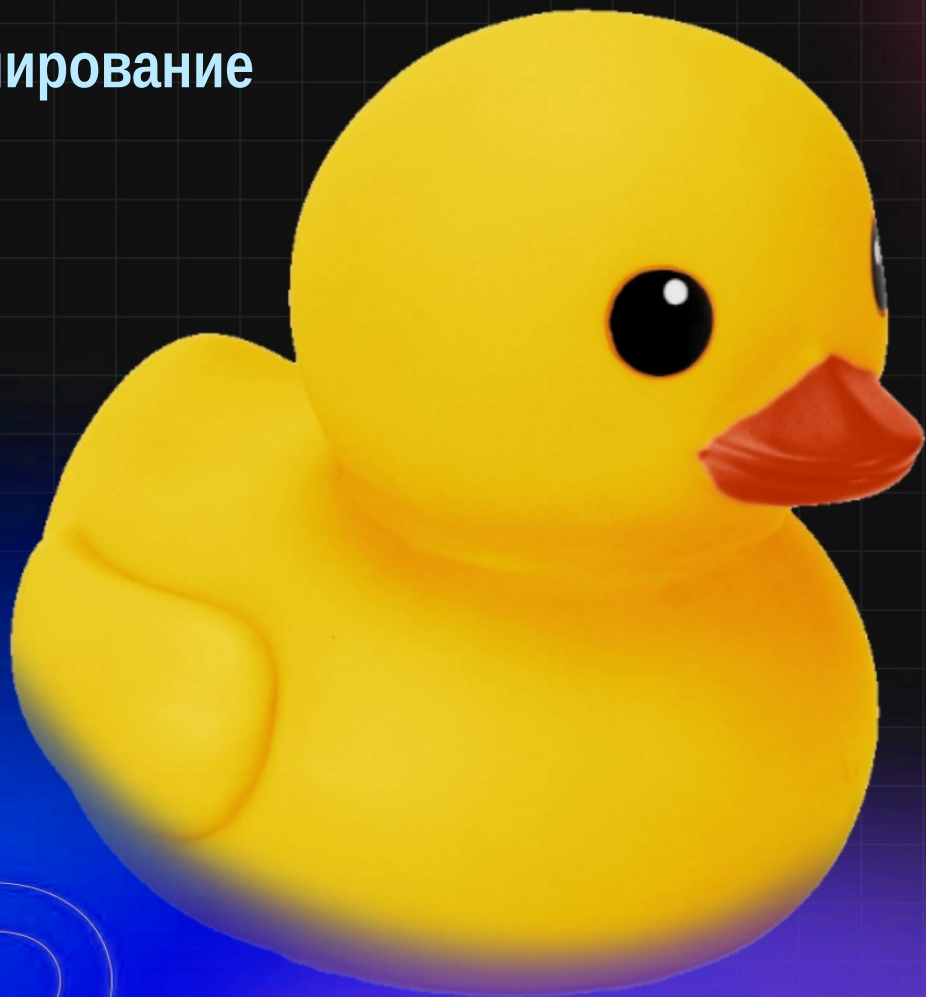
- ❖ `updateInt(String, int)`
- ❖ `updateInt(int, int)`
- ❖ `updateString(String, String)`
- ❖ `updateString(int, String)`
- ❖ `updateRow()`

- Добавление строк

- ❖ `moveToInsertRow()`
- ❖ `updateInt(String, int)`
- ❖ `insertRow()`

- `ResultSetMetaData ResultSet.getMetaData()`
  - ❖ `getTableName()`
  - ❖ `getColumnCount()`
  - ❖ `getColumnName(int n)`
  - ❖ `getColumnType(int n)`
- `DatabaseMetaData Connection.getMetaData()`
  - ❖ `getCatalogs()`
  - ❖ `getTables()`
  - ❖ `getSchemas()`

Программирование  
2 семестр  
2024



ІТМО

Расширения JDBC

- `javax.sql.DataSource`
  - ❖ Позволяет получить соединение с БД
  - ❖ `org.postgresql.ds.PGSimpleDataSource`
- `ConnectionPoolDataSource`
  - ❖ Поддержка пула соединений
  - ❖ `org.postgresql.ds.PGPoolingDataSource`
- `XADataSource`
  - ❖ Поддержка распределенных транзакций



```
import org.postgresql.ds.PGSimpleDataSource;  
  
PGSimpleDataSource ds = new PGSimpleDataSource();  
  
ds.setServerName(...);  
ds.setDatabaseName(...);  
ds.setUser(...);  
ds.setPassword(...);  
  
Connection conn = ds.getConnection();
```

```
import javax.naming.*;
```

```
Context context = new InitialContext();
```

```
DataSource ds = ...           // подготовка DataSource
```

```
context.bind("testDB", ds); // привязка к контексту
```

```
// получение DataSource из контекста
```

```
Context context = new InitialContext();
```

```
DataSource ds = (DataSource)context.lookup("testDB");
```



- `javax.rowset.*`
- Единый интерфейс для всех операций
- RowSet extends ResultSet
  - `setUrl()`,
  - `,`
  - `setPassword()`,
  - `setCommand("Select * from ...");`
  - ❖ `execute()`
  - ❖ `next()`
  - ❖ `getXXX()`

```
var factory = RowSetProvider.newFactory()  
var rowset = factory.createJdbcRowSet();
```

- RowSetFactory factory = RowSetProvider.newFactory();
  - ❖ factory.createJdbcRowSet();
  - ❖ factory.createCachedRowSet();
  - ❖ factory.createWebRowSet();
  - ❖ factory.createFilteredRowSet();
  - ❖ factory.createJoinRowSet();

- **JdbcRowSet** — простая разновидность RowSet
  - ❖ Поддерживает соединение с базой данных
  - ❖ По умолчанию:
    - Тип: SCROLL\_INSENSITIVE
    - Конкурентность: CONCUR\_UPDATABLE
    - Включено экранирование спецсимволов

- **JdbcRowSet** — простая разновидность RowSet

```
JdbcRowSet rs = factory.createJdbcRowSet();  
rs.setUrl(""); ...  
rs.setCommand("Select * from users");  
rs.execute();  
rs.last();  
rs.getInt("id");  
rs.updateString("name", "Pupkin");  
rs.UpdateRow();
```

- CachedRowSet
  - ❖ Результат запроса может кэшироваться
  - ❖ Синхронизация с базой
  - ❖ Разрешение конфликтов
  - ❖ `acceptChanges()`

- CachedRowSet

```
CachedRowSet rs = factory.createCachedRowSet();  
rs.setUrl(""); ...  
rs.setCommand("Select * from users");  
rs.execute();  
rs.last();  
rs.getInt("id");  
rs.updateString("name", "Pupkin");  
rs.acceptChanges();
```



- WebRowSet
  - ❖ Может записывать и читать результат в виде XML
  - ❖ writeXML()
  - ❖ readXML()

- WebRowSet

```
WebRowSet rs = factory.createWebRowSet();  
rs.setUrl(""); ...  
rs.setCommand("Select * from users");  
rs.execute();  
rs.writeXML(new FileWriter("data.xml"));
```

- FilteredRowSet
  - ❖ Фильтрация строк (аналог WHERE)
  - ❖ `setFilter(Predicate p)`
- Predicate
  - ❖ `boolean evaluate(Object value, int/String column)`
  - ❖ `boolean evaluate(RowSet rowset)`

- FilteredRowSet

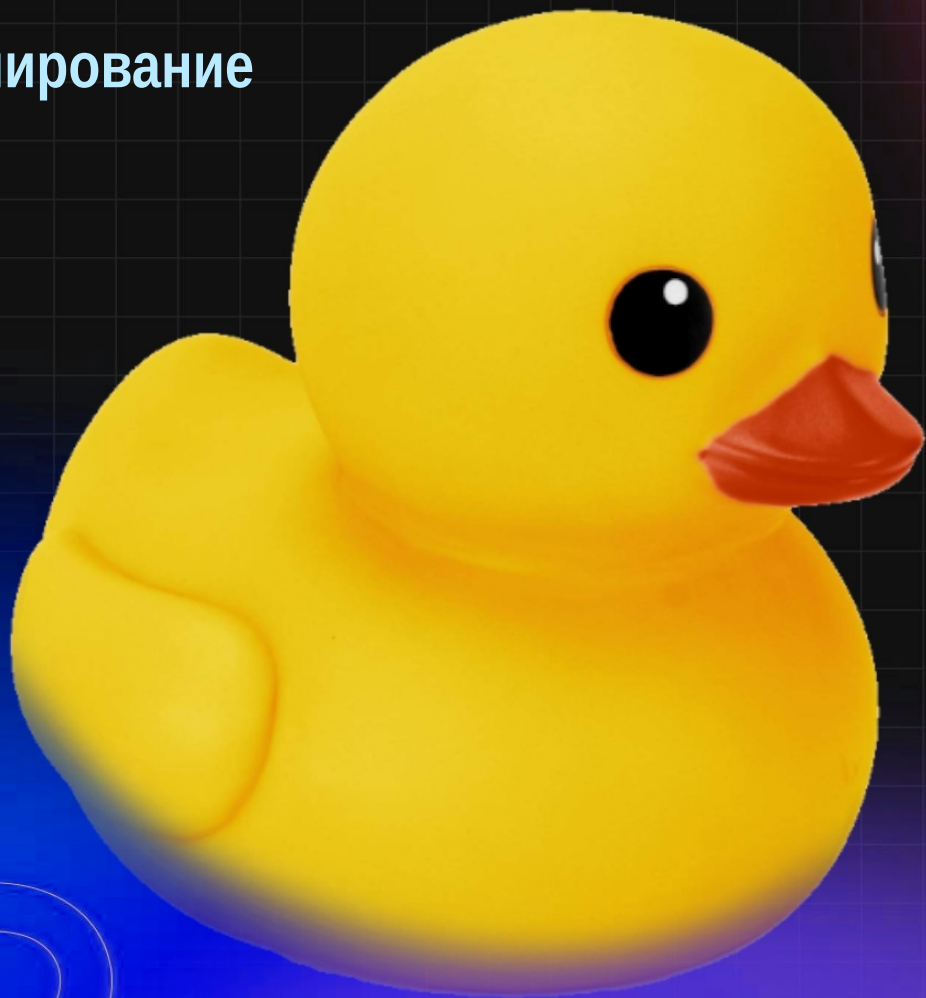
```
FilteredRowSet rs = factory.createFilteredRowSet();  
rs.setUrl(""); ...  
rs.setCommand("Select * from users");  
rs.execute();  
Predicate filter = new Predicate();  
rs.setFilter(filter);
```

- JoinRowSet
  - ❖ Соединение нескольких результатов (JOIN)
  - ❖ addRowSet()
  - ❖ setJoinType()
  - ❖ toCachedRowSet()

- JoinRowSet

```
JoinRowSet js = factory.createJoinRowSet();  
CachedRowSet users; // "Select * from users"  
CachedRowSet groups; // "Select * from groups"  
users.setMatchColumn("uid");  
groups.setMatchColumn("uid");  
js.addRowSet(users);  
js.addRowSet(groups);
```

Программирование  
2 семестр  
2024



ІТМО

Провайдеры  
служб

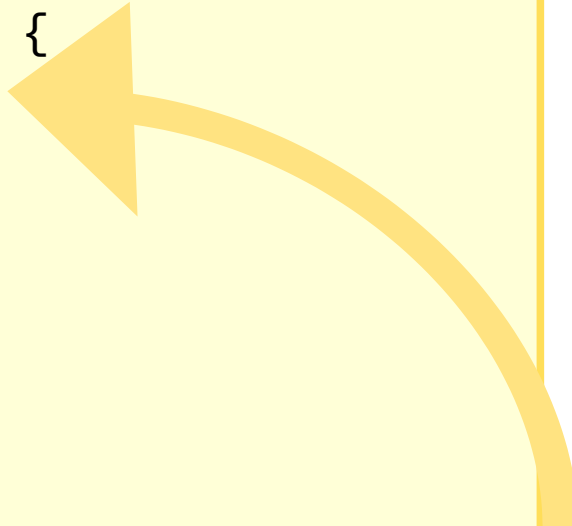
- Служба или сервис
- Разные реализации
- Нужен механизм поиска и загрузки служб



```
class CheatSheet {  
    public String getAnswer(String question) {  
        Map source =  
        String answer = source.get(question);  
        if (answer != null) return answer;  
        else return "Epic fail";  
    }  
}
```

```
class CheatSheet {  
    public String getAnswer(String question) {  
        Map source = knowledge.getSource();  
        String answer = source.get(question);  
        if (answer != null) return answer;  
        else return "Epic fail";  
    }  
}  
interface Knowledge {  
    Map<String,String> getSource();  
}
```

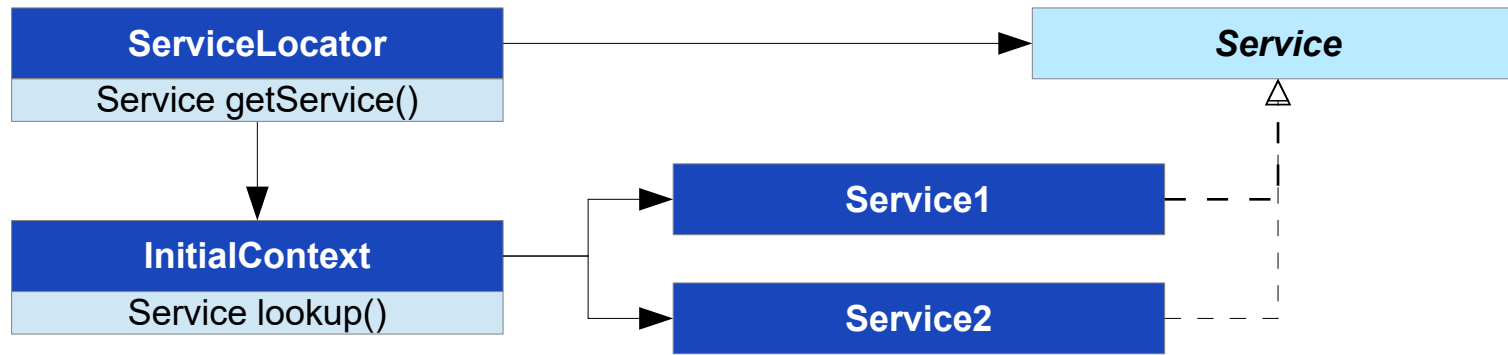
```
class CheatSheet {  
    public String getAnswer(String question) {  
        Map source = knowledge.getSource();  
        String answer = source.get(question);  
        if (answer != null) return answer;  
        else return "Epic fail";  
    }  
}  
interface Knowledge {  
    Map<String,String> getSource();  
}
```



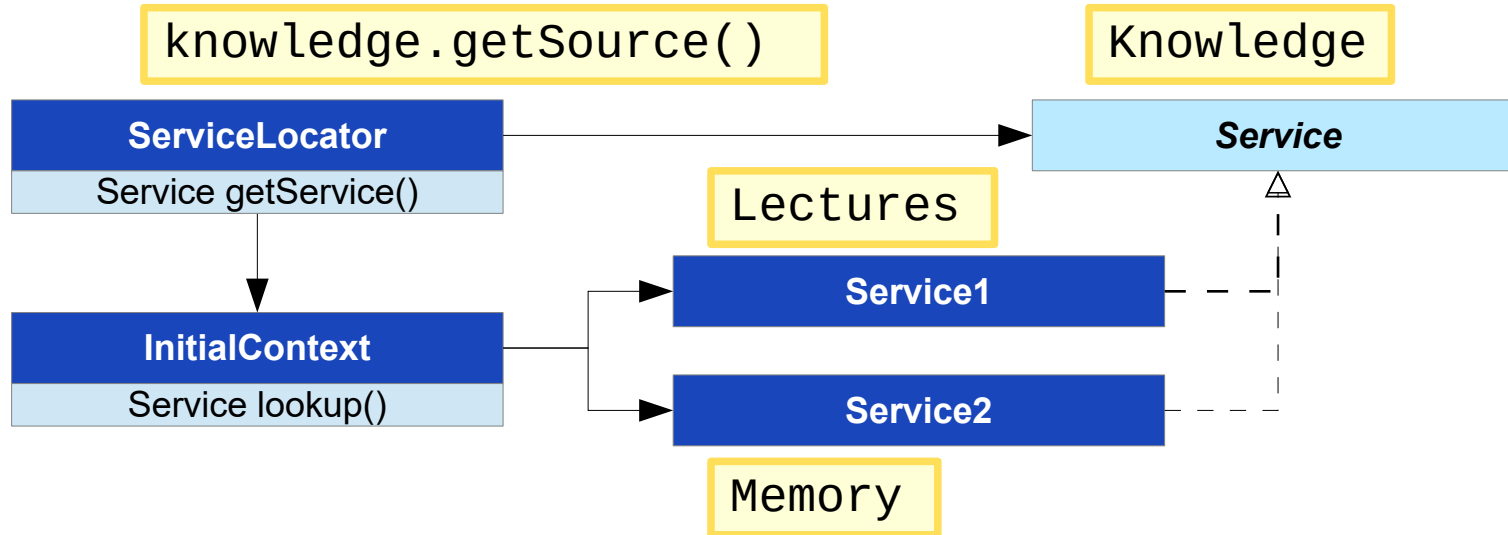
```
class Memory implements Knowledge  
class Lecture implements Knowledge  
class CallFriend implements Knowledge  
class Miracle implements Knowledge
```

# Как передать объект?

- Шаблон ServiceLocator



- Шаблон ServiceLocator



- Класс `java.util.ServiceLoader<Service>`
  - ❖ `static ServiceLoader<Service> load(Service.class)`
  - ❖ `Iterator<Service> iterator()`

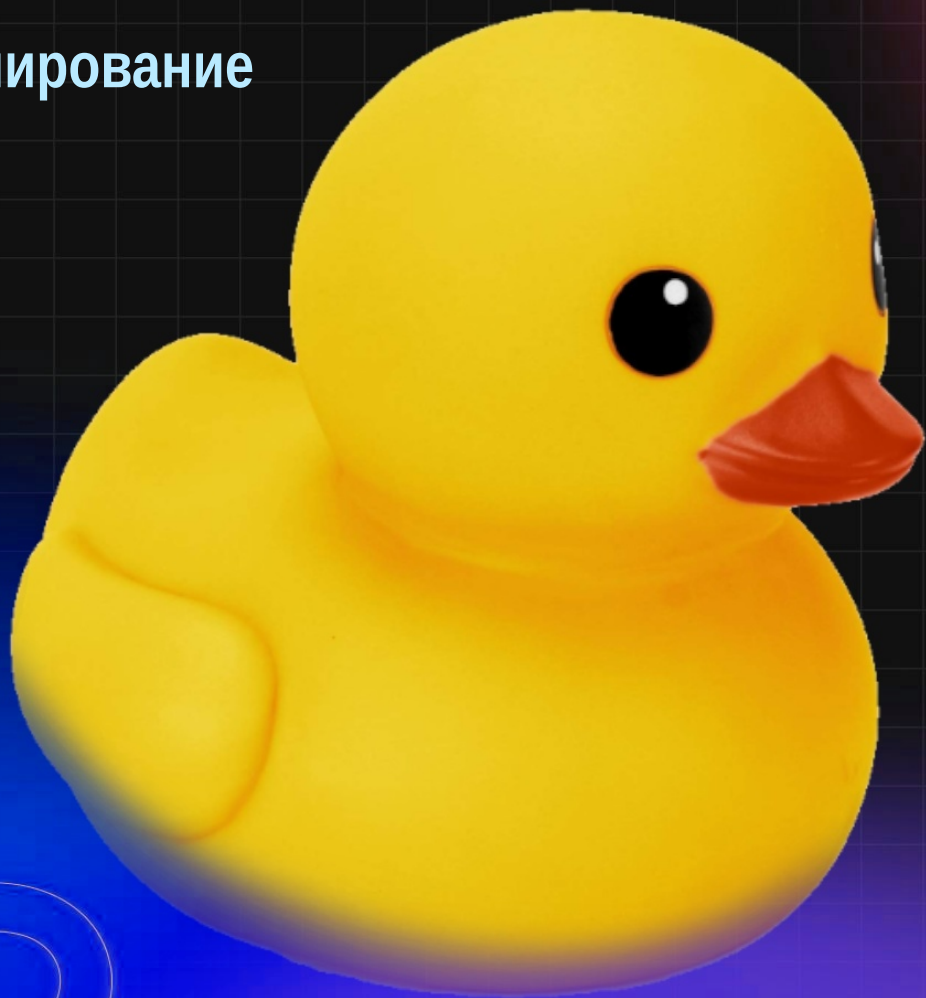
- **interface** `spi.Service`
  - ❖ `public void execute();`
- **public class** `spi.DefaultServiceImpl {`
  - ❖ **public DefaultServiceImpl() { ... }**
  - ❖ `public void execute() { ... }`
- `service.jar`
  - ❖ каталог `META-INF/services/`
    - файл `spi.Service`
      - ▶ `spi.DefaultServiceImpl`

- `java.sql.DriverManager`
- `java.nio.file.spi.FileSystemProvider`
- `java.nio.channels.spi.AsynchronousChannelProvider`
- `java.nio.channels.spi.SelectorProvider`
- `java.nio.charset.spi.CharsetProvider`
- `java.text.spi.DateTimeFormatProvider`
- `java.text.spi.NumberFormatProvider`
- `java.util.spi.CalendarDataProvider`





Программирование  
2 семестр  
2024



**ІТМО**

**Модули**

- Классы
  - ❖ `public class` - доступен всем
  - ❖ объединяются в пакеты (доступ внутри пакета)
  - ❖ упаковываются в JAR-архивы
- Пакеты ≈ каталоги
  - ❖ пакет и JAR - разные сущности
  - ❖ файл `package-info.java` / `.class`

- classpath
  - ❖ просмотр всех путей, загрузка первого найденного класса
- конфликт версий
  - ❖ не та версия класса, классы разных версий
- кастомные загрузчики классов
- нет зависимостей на уровне JVM
- JRE - Java Runtime Environment
  - ❖ rt.jar - содержит всю стандартную библиотеку

- Явные зависимости (requires и exports)
- модульная библиотека
  - ❖ базовый модуль `java.base`
  - ❖ платформенные модули (все остальные)
  - ❖ новый формат артефактов - `jmod`
- `jlink` - для сборки кастомного образа
  - ❖ `rt.jar` удален в Java 11
- модульный JAR (+`module-info.class`)

- Декларация модуля - module-info.java / .class

```
module my.mod {  
    requires java.base;  
    requires java.sql;  
    exports my.package;  
}
```

- Опции javac и java

```
javac --module-path mods:libs -d ./out  
  
java --module-path mods:libs --module my.mod
```

- `module m0` - наш модуль
- `exports p1` - экспортирует пакет `p1` всем (доступ во время компиляции к **открытым** сущностям пакета)
- `exports p1 to m1` - доступ только для модуля `m1`
- `opens p2` - открывает пакет `p2` всем (доступ с помощью рефлексии во время выполнения ко **всем** сущностям)
- `opens p2 to m2` - открыт только для модуля `m2`

- module m0 - наш модуль
- **requires** m1 - m0 требует m1 при компиляции и выполнении
- **requires transitive** m2 - m0 требует m2 и модули, зависящие от m0, тоже требуют m2
- **requires static** m3 - m0 требует m3 при компиляции, но не обязательно при выполнении

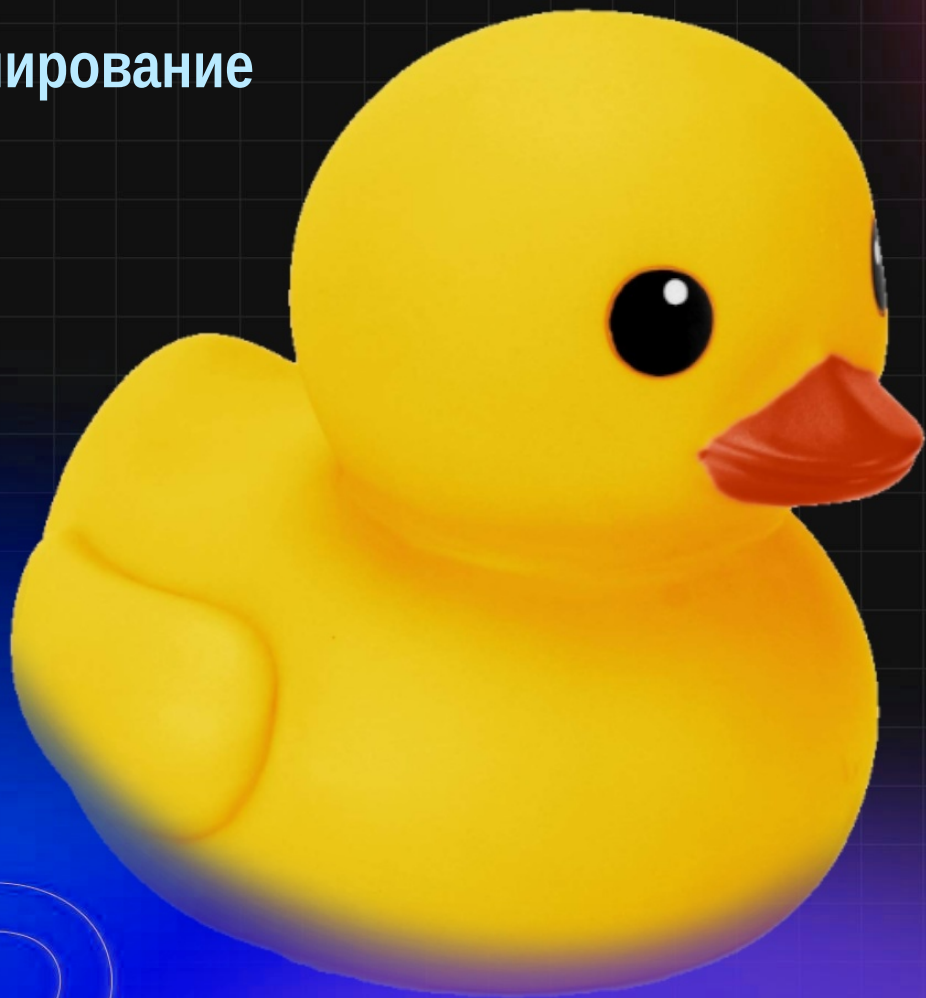
- **module m0** - наш модуль
- **uses i1** - использует сервис с интерфейсом i1 (может загружать его с помощью ServiceLoader)
- **provides i2 with c1, c2** - предоставляет интерфейс сервиса i2, который реализуют конкретные классы c1 и c2



- Безымянный модуль
  - ❖ автоэкспорт всех своих пакетов
  - ❖ имеет доступ ко всем другим модулям
  - ❖ используется classpath

Программирование  
2 семестр  
2024

ІТМО



Безопасное  
хранение паролей

- `CREATE TABLE users (varchar name, varchar password);`

Пароль нельзя хранить в открытом виде

- Доступ к базе
- Одинаковые пароли
- Резервное копирование
- SQL-инъекции и другие методы

```
CREATE TABLE users (varchar name, varchar password);  
CREATE TABLE cats (varchar cat, varchar owner);  
q1 = "SELECT * FROM users WHERE name = '%s' AND password = '%s'";  
q2 = "SELECT cat, owner FROM cats WHERE owner = '%s'";  
username = Scanner.readLine(); password = Scanner.readLine();  
statement.executeQuery(String.format(q1, username, password));  
statement.executeQuery(String.format(q2, username));
```

```
CREATE TABLE users (varchar name, varchar password);
```

```
CREATE TABLE cats (varchar cat, varchar owner);
```

```
q1 = "SELECT * FROM users WHERE name = '%s' AND password = '%s'";
```

```
q2 = "SELECT cat, owner FROM cats WHERE owner = '%s'";
```

Name:

Password:

```
SELECT * FROM users WHERE name = 'Simon' AND password = '123456';
```

```
SELECT cat FROM cats WHERE owner = 'Simon';
```

```
CREATE TABLE users (varchar name, varchar password);
```

```
CREATE TABLE cats (varchar cat, varchar owner);
```

```
q1 = "SELECT * FROM users WHERE name = '%s' AND password = '%s'";
```

```
q2 = "SELECT cat, owner FROM cats WHERE owner = '%s'";
```

Name:

Password:

```
SELECT * FROM users WHERE name = 'Simon'; -- ' AND password = '1';
```

Login OK!

```
CREATE TABLE users (varchar name, varchar password);
```

```
CREATE TABLE cats (varchar cat, varchar owner);
```

```
q1 = "SELECT * FROM users WHERE name = '%s' AND password = '%s'";
```

```
q2 = "SELECT cat, owner FROM cats WHERE owner = '%s'";
```

Name: ' OR ''='

Password: ' OR ''='

```
SELECT * FROM users WHERE name = '' OR ''=' AND password = '' OR ''='
```

Login OK!

```
CREATE TABLE users (varchar name, varchar password);
```

```
CREATE TABLE cats (varchar cat, varchar owner);
```

```
q1 = "SELECT * FROM users WHERE name = '%s' AND password = '%s'";
```

```
q2 = "SELECT cat, owner FROM cats WHERE owner = '%s'";
```

Name: `' UNION SELECT name,password FROM users WHERE name<>'`

```
SELECT cat, owner FROM cats WHERE owner = ''
```

```
UNION SELECT name,password FROM users WHERE name<>'';
```



```
CREATE TABLE users (varchar name, varchar password);
```

```
CREATE TABLE cats (varchar cat, varchar owner);
```

```
q1 = "SELECT * FROM users WHERE name = '%s' AND password = '%s'";
```

```
q2 = "SELECT cat, owner FROM cats WHERE owner = '%s'";
```

Name: `' UNION SELECT name,password FROM users WHERE name<>'`

```
SELECT cat, owner FROM cats WHERE owner = ''
```

```
UNION SELECT name,password FROM users WHERE name<>'';
```

```
Simon's Cat      Simon
```

```
admin             @#p1nk_m0nkey$%
```



- Вместо пароля — хеш пароля
- Функция **необратимая**
- Минимальное число **коллизий**
- MD2, MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512
- `md5("hello") = 5d41402abc4b2a76b9719d911017c592`

- Дни рождения
- В группе  $> 23$  человек
- Вероятность совпадения дня рождения?

- Дни рождения



- Вместо пароля — хеш пароля
- Функция **необратимая** — **но можно подобрать**
- Минимальное число **коллизий** — **тоже можно подобрать**
- MD2, MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512
- `md5("hello") = 5d41402abc4b2a76b9719d911017c592`
- Словарные атаки

<https://md5.gromweb.com/>

<http://reversemd5.com/>

# Добавим к хешу соль

- Вместо пароля — хеш пароля + соль
- Соль — некая случайная последовательность



`md5("hello" + "$!ns50D") = fcac10c41f9f67090e450161db4bca5a`

`md5("hello" + "HOxc3@") = a893f9063e4314635e05f8c46b547f7a`

# Добавим к хешу соль

- Вместо пароля — хеш пароля + соль
- Соль — некая случайная последовательность



`md5("hello" + "$!ns50D") = fcac10c41f9f67090e450161db4bca5a`

`md5("hello" + "HOxc3@") = a893f9063e4314635e05f8c46b547f7a`

- Храним пароль + соль + алгоритм + сложность
- Все равно — "hello", "12345" — словарные атаки

# Добавим перец

- В базе храним хеш пароля + соль
- Перец храним в приложении (отдельно)
- `md5("hello" + "kFz<Q%ps" + "$Ins50D")`





```
java.security.MessageDigest
```

```
MessageDigest md = MessageDigest.getInstance("MD5");  
String user      = Console.readLine()  
String passwd    = Console.readPassword();  
String salt      = getRandomString();  
String pepper    = "*63&^mVLC(#"  
byte[] hash = md.digest(  
    (passwd + pepper + salt).getBytes("UTF-8"));  
  
insert into users (user, salt, hash);
```

- Входящие пароли - хеш, соль
- Исходящие пароли - отдельно от кода!
  - ❖ Запрашивать при старте
  - ❖ Конфигурационный файл с ограничением прав
  - ❖ Файлы свойств (Properties)
  - ❖ Переменная окружения
  - ❖ \$HOME/.pgpass