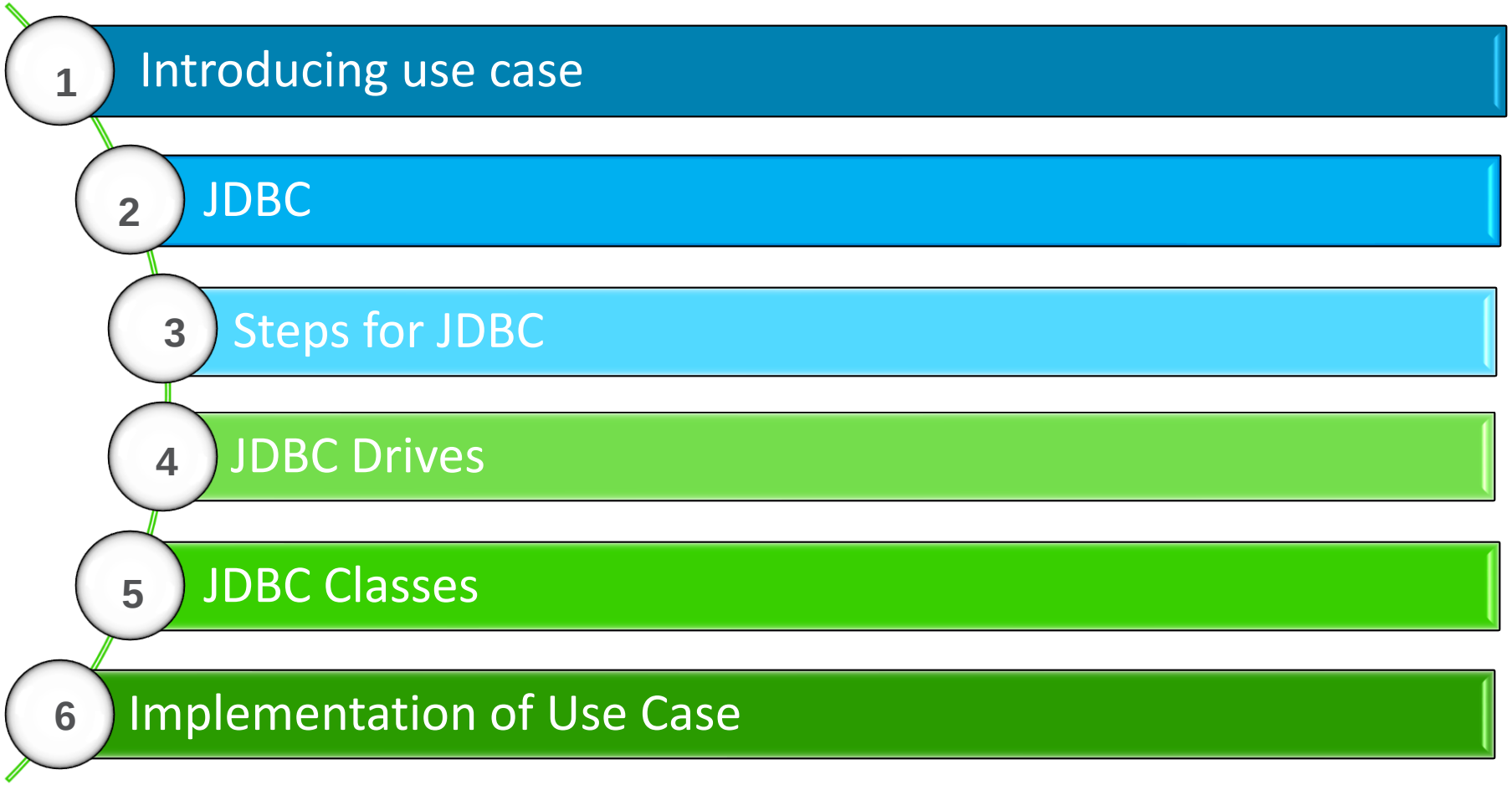
axess academy

Java with Postgres

axess academy

Modules



axess academy

Introducing the Use Case

Bank Customer Management System.

axess academy

- Persist the customer data into tables using JDBC.
- Save the customer details into database as and when new customer gets added.
- Delete the customer details from database as and when admin requests to delete any customer.

Bank Customer Management System

axess academy

- Display the menu to user

```
Welcome to Standard Chartered Bank
Please enter your choice
1 for Add new Customer
2 for Display Customers
3 for Search Customer
4 for Delete Customer
5 for Exit the bank application
```

- Create the customer class having fields cutsomerid as int, name as string, mailid as string contact as string and account type as string.
- If user enters 1 , please provide the details. On valid entries these details should be added into the database with auto generated customerid.

```
Please enter customer details :
Enter name :
Sandra
Enter email :
Sandra@gmail.com
Enter contact :
9988778877
Enter account type (Savings or Current):
Savings
Customer added suucessfully with customer id 4656
```

Bank Customer Management System …

axess academy

- While adding the customer, make sure admin provides valid entries like name can only have alphabets, email should be valid one , contact no should have 10 digits and account type can be either Savings or current. If any one of the condition fails, the customer should not be added in the database and display the appropriate error message.
- If user enters 2 , display all existing customer details using collection.

```
Customer Id = 7262, Customer name = Sandra, Customer email = Sandra@gmail.com, Customer conta
Customer Id = 1014, Customer name = Michelle, Customer email = Michelle@gmail.com, Customer c
Customer Id = 2680, Customer name = Steve, Customer email = Steve@gmail.com, Customer contact
```

- If user enters 3 , display the matched customer details.

```
3
Please enter customer id
1014
Customer Id = 1014, Customer name = Michelle, Customer email = Michelle@gmail.com, Customer
```

- If user enters 4 , delete the matched customer details.

```
4
Please enter customer id to be deleted
1014
Deleted customer with id =1014
```

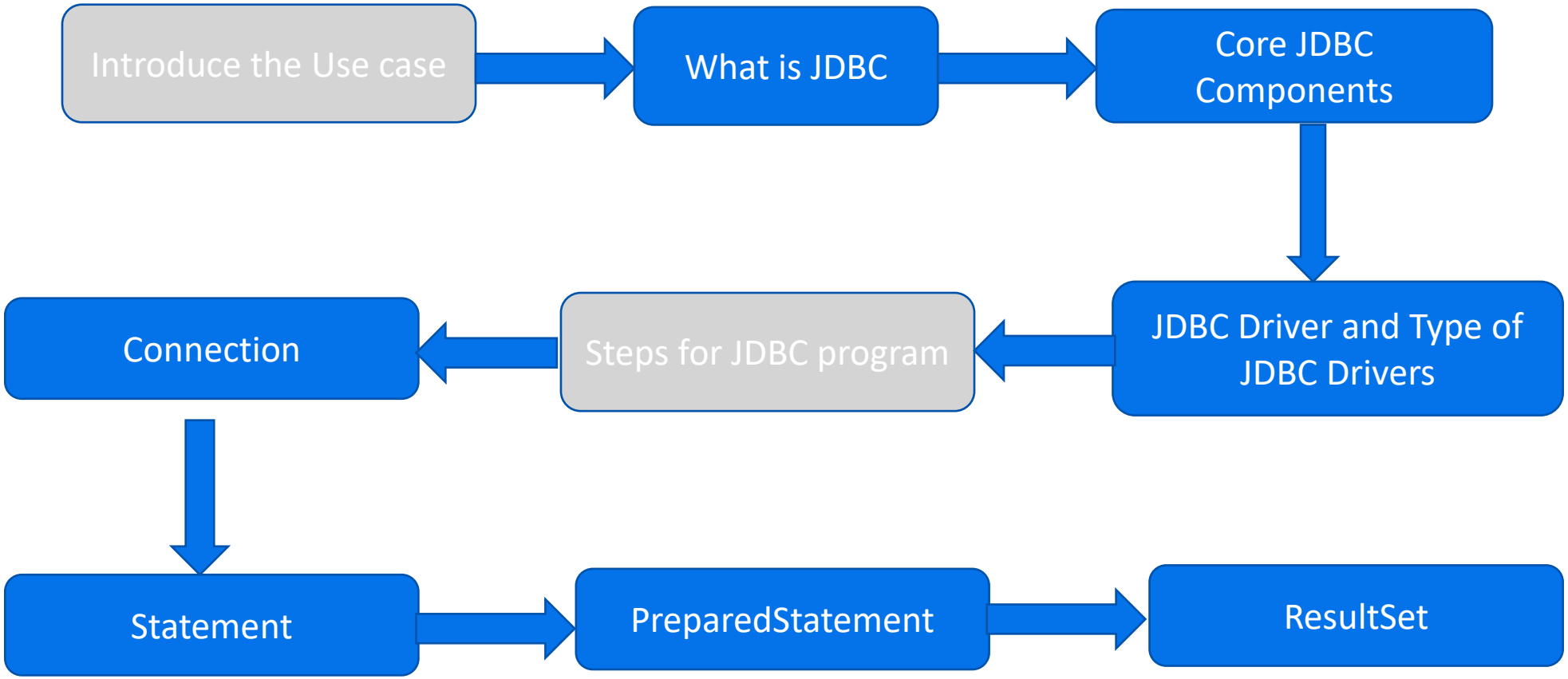
- If user enters 5, exit the application.

axess academy

JDBC (Java Data Base Connectivity)

Module 5 : JDBC

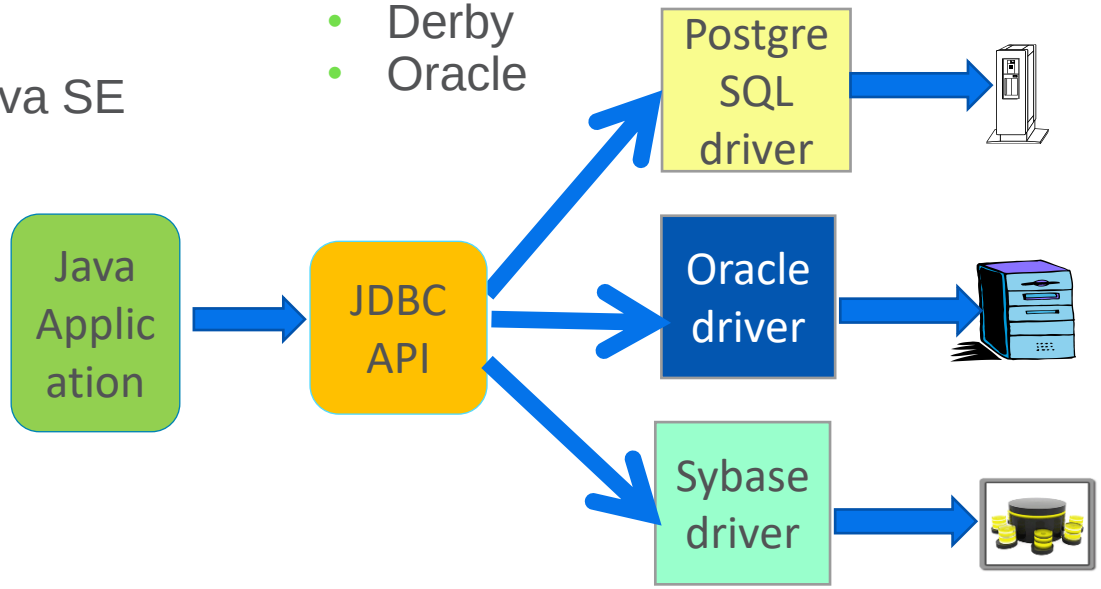
axess academy



JDBC

- The Java *JDBC* API (Java Database Connectivity) enables Java applications to connect to relational databases like MySQL, PostgreSQL, MS SQL Server, Oracle, H2 Database etc.
- The JDBC API makes it possible to query and update relational databases, as well as call stored procedures, and obtain meta data about the database.
- The Java JDBC API is part of the core Java SE SDK, making JDBC available to all Java applications that want to use it.

- Here is a list of popular open source and commercial relational databases which have JDBC drivers so you can use them from Java:
  - Microsoft SQL Server
  - H2Database
  - MariaDB
  - PostgreSQL
  - My SQL
  - Derby
  - Oracle



Core JDBC Components

axess academy

- The JDBC API consists of the following core parts:
  - **JDBC Drivers**: A JDBC driver is a collection of Java classes that enables you to connect to a certain database. For instance, MySQL will have its own JDBC driver. A JDBC driver implements a lot of the JDBC interfaces. When your code uses a given JDBC driver, it actually uses the standard JDBC interfaces. The concrete JDBC driver used is hidden behind the JDBC interfaces. Thus you can plugin a new JDBC driver without your code noticing it.

  - **Connections**: Once a JDBC driver is loaded and initialized, you need to connect to the database. You do so by obtaining a connection to the database via the JDBC API and the loaded driver. All communication with the database happens via a connection. An application can have more than one connection open to a database at a time. This is actually very common.

  - **Statements**: A statement is what you use to execute queries and updates against the database. There are a few different types of statements like PreparedStatement, CallableStatement you can use. Each statement corresponds to a single query or update.

  - **Result Sets** : When you perform a query against the database you get back a ResultSet . You can then traverse this ResultSet to read the result of the query.

JDBC Steps

axess academy

- Register Driver

```
// loads driver and register with driver manager
 Class.forName("org.postgresql.Driver");
```

- Get Connection

```
private final String url = "jdbc:postgresql://localhost:8004/dbname";
private final String user = "postgres";
private final String password = "postgres";
```

```
Connection conn = DriverManager.getConnection(url, user, password);
```

- Create Statement

```
Statement stmt = conn.createStatement();
```

- Execute Query

```
ResultSet rs = stmt.executeQuery( "SELECT * FROM COMPANY");
while ( rs.next() ) {
   int id = rs.getInt("id");
   String  name = rs.getString("name");
   int age  = rs.getInt("age");
   String  address = rs.getString("address");
   float salary = rs.getFloat("salary");
 }
```

- Close connection

```
// close connection
stmt.close();
rs.close();
conn.close();
```

Connection

- A Connection is the session between java application and database. The Connection interface is a factory of Statement, PreparedStatement.

- Commonly Used methods are:
  1. **public Statement createStatement():** creates a statement object that can be used to execute SQL queries.
  2. **public PrepareStatement prepareSatement(String sql):** creates a prepare statement object that can be used to execute SQL queries.
  3. **public void setAutoCommit(boolean status):** is used to set the commit status.By default it is true.
  4. **public void commit():** saves the changes made since the previous commit/rollback permanent.
  5. **public void rollback():** Drops all changes made since the previous commit/rollback.
  6. **public void close():** closes the connection and Releases a JDBC resources immediately.

```
private final String url = "jdbc:postgresql://localhost:8004/dbname";
private final String user = "postgres";
private final String password = "postgres";
```

```
Connection conn = DriverManager.getConnection(url, user, password);
```

axess academy

Statement

- The **Statement interface** provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.
- The commonly used methods are:
  1. **public ResultSet executeQuery(String sql):** is used to execute SELECT query. It returns the object of ResultSet.
  2. **public int executeUpdate(String sql):** is used to execute specified query, it may be insert, update, delete etc.
  3. **public boolean execute(String sql):** is used to execute queries that may return multiple results.

```
String sql = "select cust_id from customers where cust_id="+customerId+"";
Statement stmt= con.createStatement();
ResultSet rs=stmt.executeQuery(sql);
while (rs.next()) {
    System.out.println(rs.getInt(1));
}
```

axess academy

PreparedStatement

- The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.
- The performance of the ap
```
String sql = "insert into customers values(?,?,?,?,?)";
```
because query is compiled only once.
- The commonly used methods are:
  1. **public ResultSet executeQuery():** is used to execute SELECT query. It returns the object of ResultSet.
  2. **public int executeUpdate():** is used to execute specified query, it may be insert, update, delete etc.
  3. **public boolean execute():** is used to execute queries that may return multiple results.

```
String sql = "insert into customers values(?,?,?,?,?)";
PreparedStatement pstmt = con.prepareStatement(sql);
pstmt.setInt(1, customer.getCustomerId());
pstmt.setString(2, customer.getName());
pstmt.setString(3, customer.getContact());
pstmt.setString(4, customer.getEmail());
pstmt.setString(5, customer.getAccountType());
pstmt.executeUpdate();
```

ResultSet

- The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row.
- The commonly used methods are:
  1. **public boolean next():** is used to move the cursor to the one row next from the current position.
  2. **public boolean previous():** is used to move the cursor to the one row previous from the current position.
  3. **public int getInt(int columnIndex):** is used to return the data of specified column index of the current row as int.
  4. **public int getInt(String columnName):** is used to return the data of specified column name of the current row as int.

```
PreparedStatement pstmt = con.prepareStatement(sql);
ResultSet rs = pstmt.executeQuery();
while (rs.next()) {
    Customer customer = new Customer();
    customer.setCustomerId(rs.getInt(1));
    customer.setName(rs.getString(2));
    customer.setContact(rs.getString(3));
    customer.setEmail(rs.getString(4));
    customer.setAccountType(rs.getString(5));
}
```

Types of JDBC Driver

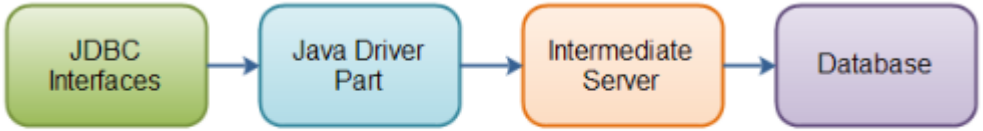
- There are four different JDBC driver types. Today most of the drivers are Type 4.
- These driver types are:
  1. Type 1: JDBC-ODBC bridge JDBC driver

Type 1 JDBC driver.

  2. Type 2: Java + Native code JDBC driver

Type 2 JDBC driver.

  3. Type 3: All Java + Middleware translation JDBC driver



Type 3 JDBC driver.

  4. Type 4: All Java JDBC driver.

Type 4 JDBC driver.

axess academy