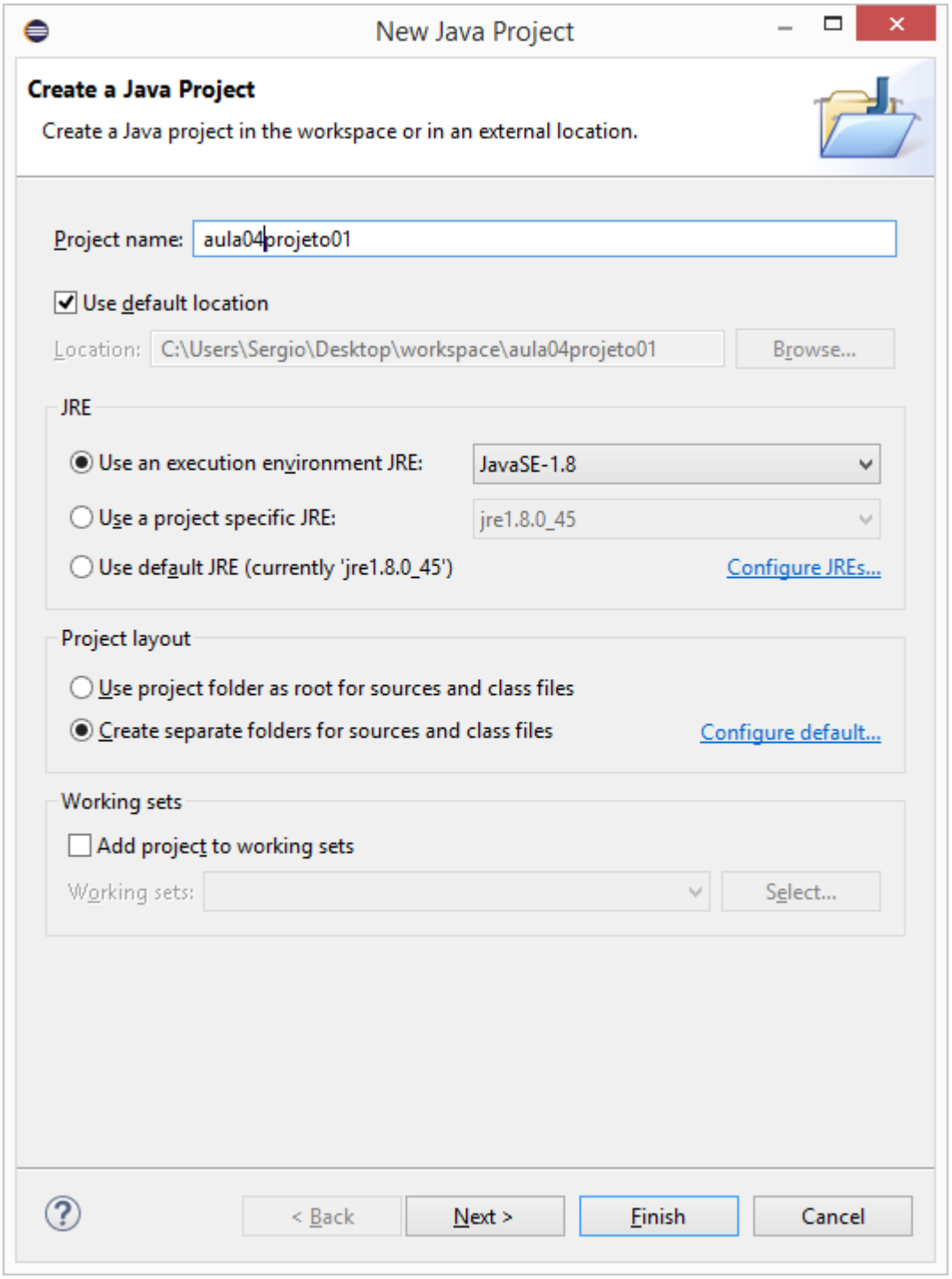
Novo projeto:

File > New > Java Project



Criando o script da base de dados
Linguagem SQL

- script.sql

```
drop database if exists aula04;
create database aula04;
use aula04;

create table cliente(
      idcliente       integer              auto_increment,
      nome            varchar(50)          not null,
      email           varchar(50)          not null unique,
      primary key(idcliente));

desc cliente;

insert into cliente(nome, email) values('Ana', 'ana@gmail.com');
insert into cliente(nome, email) values('Rui', 'rui@gmail.com');
insert into cliente(nome, email) values('Leo', 'leo@gmail.com');
insert into cliente(nome, email) values('Max', 'max@gmail.com');
insert into cliente(nome, email) values('Bia', 'bia@gmail.com');

select * from cliente;
```

Criando a entidade Cliente (JavaBean)
entities/Cliente.java

```
package entities;

public class Cliente implements Comparable<Cliente>{

      // atributos..
      private Integer idCliente;
      private String nome;
      private String email;

      public Cliente() {
            // Construtor default..
      }

      // sobrecarga de construtores..
      public Cliente(Integer idCliente, String nome, String email) {
            this.idCliente = idCliente;
            this.nome = nome;
            this.email = email;
      }
```

```
	public Integer getIdCliente() {
		return idCliente;
	}

	public void setIdCliente(Integer idCliente) {
		this.idCliente = idCliente;
	}

	public String getNome() {
		return nome;
	}

	public void setNome(String nome) {
		this.nome = nome;
	}

	public String getEmail() {
		return email;
	}

	public void setEmail(String email) {
		this.email = email;
	}

	@Override
	public int hashCode() {
		return idCliente.hashCode();
	}

	@Override
	public boolean equals(Object obj) {
		if(obj instanceof Cliente){
			Cliente c = (Cliente) obj;
			if(c.getIdCliente() != null){
				return c.getIdCliente().equals(idCliente);
			}
		}
		return false;
	}

	@Override
	public String toString() {
		return idCliente + ", " + nome + ", " + email;
	}

	@Override
	public int compareTo(Cliente c) {
		return idCliente.compareTo(c.getIdCliente());
	}
}
```

Criando a Classe de conexão com o banco de dados
DAO – Data Access Object

Objeto de acesso a dados (ou simplesmente DAO, acrônimo de Data Access Object), é um padrão para persistência de dados que permite separar regras de negócio das regras de acesso a banco de dados. Numa aplicação que utilize a arquitetura MVC, todas as funcionalidades de bancos de dados, tais como obter as conexões, mapear objetos Java para tipos de dados SQL ou executar comandos SQL, devem ser feitas por classes DAO.

```
package persistence;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

//Data Access Object -> padrão para classes de acesso a banco de dados
//JDBC - tecnologia Java utilizando para conexão com base de dados
//Java Database Connectivity
public class Dao {

	//atributos (constantes -> static final)
	private static final String DRIVER = "com.mysql.jdbc.Driver";
	private static final String URL = "jdbc:mysql://localhost:3306/aula04";
	private static final String USER = "root";
	private static final String PASSWORD = "brqbrq";

	//Principais APIs (Classes e Interfaces do JDBC)
	protected Connection con; //realiza a conexão com o banco de dados..
	protected PreparedStatement stmt; //executar comandos SQL..
	protected CallableStatement call; //executar storedprocedures..
	protected ResultSet rs;		//ler resultados de consultas..

	//métodos para abrir e fechar conexão com o banco de dados..
```

```
        protected void openConnection() throws Exception{
                //carregar o caminho do driver do banco de dados...
                Class.forName(DRIVER);
                //abrindo conexão com o banco de dados..
                con = DriverManager.getConnection(URL, USER, PASSWORD);
        }

        protected void closeConnection() throws Exception{
                if(con != null){ //se ha instancia de conexão..
                        con.close(); //fechar conexão..
                }
        }
}
```

Criando a Classe de persistência para a entidade Cliente:
/persistence/ClienteDao.java

```
package persistence;

import java.util.ArrayList;
import java.util.List;

import entities.Cliente;

public class ClienteDao extends Dao{

        //método para inserir um cliente na base de dados..
        public void insert(Cliente c) throws Exception{

                openConnection(); //abrir conexão..

                stmt = con.prepareStatement("insert into cliente
                                                (nome, email) values(?,?)");
                stmt.setString(1, c.getNome());
                stmt.setString(2, c.getEmail());
                stmt.execute();
                stmt.close();

                closeConnection(); //fechar conexão..
        }
```

```
//método para atualizar um cliente na base de dados..
public void update(Cliente c) throws Exception{

        openConnection();

        stmt = con.prepareStatement("update cliente set nome=?,
                                     email=? where idcliente=?");
        stmt.setString(1, c.getNome());
        stmt.setString(2, c.getEmail());
        stmt.setInt(3, c.getIdCliente());
        stmt.execute();
        stmt.close();

        closeConnection();
}

//método para excluir 1 cliente na base de dados..
public void delete(Integer idCliente) throws Exception{

        openConnection();

        stmt = con.prepareStatement("delete from cliente where idcliente=?");
        stmt.setInt(1, idCliente);
        stmt.execute();
        stmt.close();

        closeConnection();
}


//método para listar todos os clientes do banco de dados..
public List<Cliente> findAll() throws Exception{

        openConnection();

        stmt = con.prepareStatement("select * from cliente");
        rs = stmt.executeQuery();

        //declarando a lista de clientes..
        List<Cliente> lista = new ArrayList<Cliente>();

        //percorrer o ResultSet..
        while(rs.next()){ //enquanto tem registro..
```

```
                Cliente c = new Cliente(); //entidade..
                c.setIdCliente(rs.getInt("idcliente"));
                c.setNome(rs.getString("nome"));
                c.setEmail(rs.getString("email"));

                //adicionar na lista..
                lista.add(c); //incluindo na lista..
            }

            stmt.close(); //fechando statement
            closeConnection(); //fechar conexão..

            return lista; //retornar a lista..
        }


        //método para buscar 1 cliente pelo id..
        public Cliente findById(Integer idCliente) throws Exception{

            openConnection();

            stmt = con.prepareStatement("select * from cliente where idcliente=?");
            stmt.setInt(1, idCliente);
            rs = stmt.executeQuery();

            Cliente c = null; //sem espaço de memória..

            //se registro foi encontrado..
            if(rs.next()){

                c = new Cliente(); //instanciando o cliente..
                c.setIdCliente(rs.getInt("idcliente"));
                c.setNome(rs.getString("nome"));
                c.setEmail(rs.getString("email"));
            }

            stmt.close(); //fechando o statement..
            closeConnection(); //fechando conexão..

            return c; //retornar o cliente..
        }

}
```

A vantagem de usar objetos de acesso a dados é a separação simples e rigorosa entre duas partes importantes de uma aplicação que não devem e não podem conhecer quase que nada uma da outra, e que podem evoluir frequentemente e independentemente. Alterar a lógica de negócio podem esperar apenas a implementação de uma interface, enquanto que modificações na lógica de persistência não alteram a lógica de negocio, desde que a interface entre elas não seja modificada.

- Pode ser usada em uma vasta porcentagem de aplicações;
- Esconde todos os detalhes relativos a armazenamento de dados do resto da aplicação;
- Atua como um intermediário entre a aplicação e o banco de dados;
- Mitiga ou resolve problemas de comunicação entre a base de dados e a aplicação, evitando estados inconsistentes de dados.

No contexto específico da linguagem de programação Java, um objeto de acesso a dados como padrão de projeto de software pode ser implementado de várias maneiras. Pode variar desde uma simples interface que separa partes de acesso a dados da lógica de negócio de uma aplicação até frameworks e produtos comerciais específicos.

Criando uma classe para leitura dos dados do Cliente:

```
package input;

import java.util.Scanner;

public class InputCliente {

	// atributo..
	private Scanner scanner;

	// construtor..
	public InputCliente() {
		// inicializando..
		scanner = new Scanner(System.in);
	}

	// método para ler o id do cliente..
```

```
	public Integer lerIdCliente() {
		System.out.print("Informe o Id do Cliente......: ");
		return Integer.parseInt(scanner.nextLine());
	}

	// método para ler o id do cliente..
	public String lerNome() {
		System.out.print("Informe o Nome do Cliente....: ");
		return scanner.nextLine();
	}

	// método para ler o id do cliente..
	public String lerEmail() {
		System.out.print("Informe o Email do Cliente...: ");
		return scanner.nextLine();
	}

}
```

Criando um ENUM para gerar o menu de clientes:

```
package control.types;

public enum Menu {

	CADASTRAR,
	ATUALIZAR,
	EXCLUIR,
	CONSULTAR,
	SAIR
}
```

Criando a classe de controle para gerenciar as operações de CRUD com clientes:

```
package control;

import java.util.Scanner;

import control.types.Menu;
import entities.Cliente;
import input.InputCliente;
import persistence.ClienteDao;

public class ControleCliente {
```

```
// atributo para a classe de entrada de dados..
private InputCliente input; // null

// construtor..
public ControleCliente() {
	input = new InputCliente(); // instanciando..
}

// método para executar o cadastro de um cliente..
private void cadastrarCliente() {
	try {

		Cliente c = new Cliente(); // entidade.
		c.setNome(input.lerNome());
		c.setEmail(input.lerEmail());

		ClienteDao d = new ClienteDao();
		d.insert(c); // cadastrando.

		System.out.println("Cliente cadastrado com sucesso.");
	} catch (Exception e) {
		System.out.println("Erro: " + e.getMessage());
	}
}

// método para executar a edição de um cliente..
private void atualizarCliente() {
	try {

		ClienteDao d = new ClienteDao(); //persistencia..

		//buscar o cliente pelo id..
		Cliente c = d.findById(input.lerIdCliente());

		//verificar se o cliente foi encontrado..
		if(c != null){

			c.setNome(input.lerNome());
			c.setEmail(input.lerEmail());

			d.update(c); //atualizando..

			System.out.println("Cliente atualizado com sucesso.");
		}
```

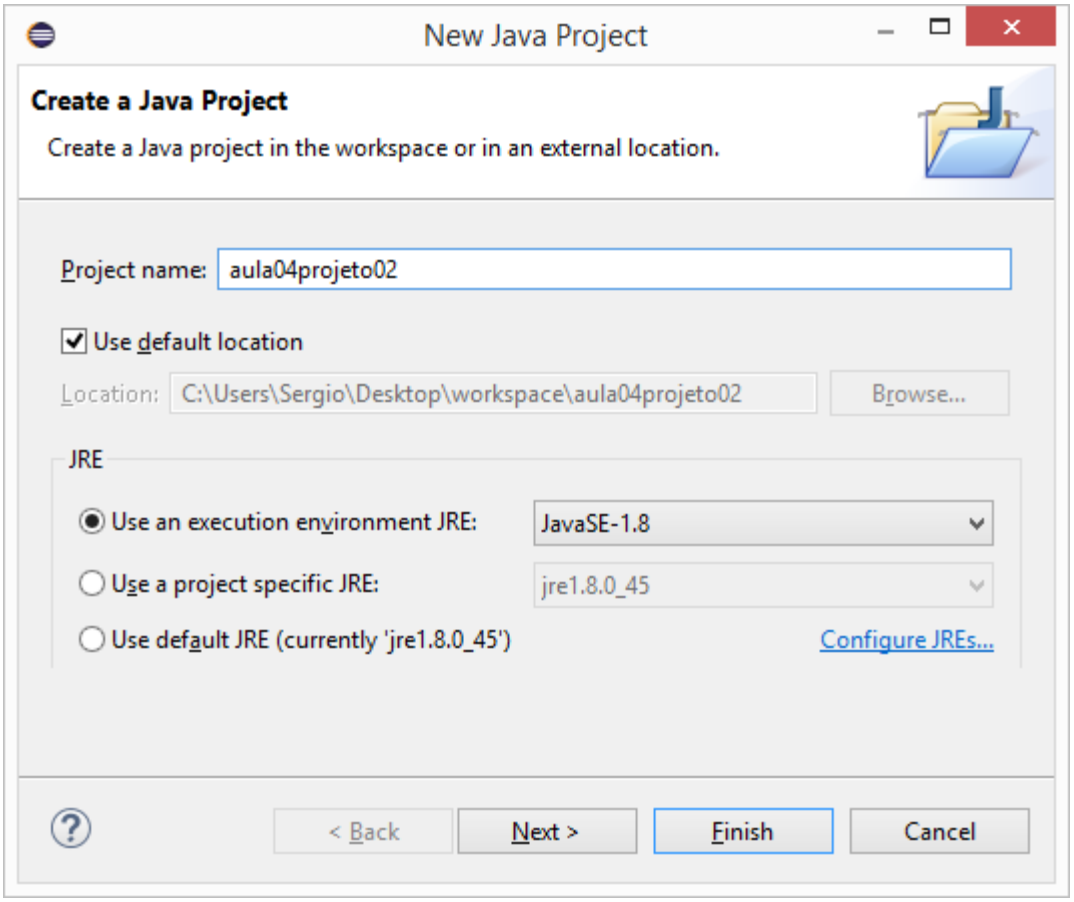
```
                else{
                        throw new Exception("Cliente não encontrado.");
                }
        } catch (Exception e) {
                System.out.println("Erro: " + e.getMessage());
        }
}

//método para excluir o cliente..
private void excluirCliente(){

        try{

                //buscar o cliente pelo id..
                ClienteDao d = new ClienteDao();
                Cliente c = d.findById(input.lerIdCliente());

                if(c != null){ //cliente foi encontrado.
                        d.delete(c.getIdCliente());

                        System.out.println("Cliente excluido com sucesso: " + c);
                }
                else{
                        throw new Exception("Cliente não encontrado.");
                }
        }
        catch(Exception e){
                System.out.println("Erro: " + e.getMessage());
        }
}

//método para imprimir todos os clientes..
private void consultarClientes(){

        try{

                ClienteDao d = new ClienteDao(); //persistencia..
                //varrendo a lista de clientes obtida do banco..
                for(Cliente c : d.findAll()){
                        System.out.println("Cliente: " + c); //toString()
                }

        }
```

```
                catch(Exception e){
                        System.out.println("Erro: " + e.getMessage());
                }
        }

        //Menu..
        public void manterClientes(){

                System.out.println("\nManter Clientes:");
                System.out.println("\tCADASTRAR,");
                System.out.println("\tATUALIZAR,");
                System.out.println("\tEXCLUIR,");
                System.out.println("\tCONSULTAR,");
                System.out.println("\tSAIR");

                System.out.print("Digite a operação desejada: ");
                Menu menu = Menu.valueOf(new Scanner(System.in)
                                .nextLine().toUpperCase());

                switch(menu){
                        case CADASTRAR:
                                cadastrarCliente();
                                break;
                        case ATUALIZAR:
                                atualizarCliente();
                                break;
                        case EXCLUIR:
                                excluirCliente();
                                break;
                        case CONSULTAR:
                                consultarClientes();
                                break;
                        case SAIR:
                                System.out.println("\nFIM DO PROGRAMA.");
                                break;
                }
        }

}
```

Executando na MAIN:

```
package principal;

import control.ControleCliente;

public class Main {

	public static void main(String[] args) {

		ControleCliente c = new ControleCliente();
		c.manterClientes();
	}
}
```

Novo projeto:



Criando o script da base de dados:

```
drop database if exists aula04b;
create database aula04b;
use aula04b;

create table funcionario(
      idfuncionario           integer           auto_increment,
      nome                    varchar(50)       not null,
      salario                 double            not null,
      dataadmissao            date              not null,
      primary key(idfuncionario));

create table endereco(
      idendereco              integer           auto_increment,
      logradouro              varchar(100)      not null,
      cidade                  varchar(50)       not null,
      estado                  varchar(50)       not null,
      cep                     varchar(15)       not null,
      idfuncionario           integer           not null unique,
      primary key(idendereco),
      foreign key(idfuncionario)
            references funcionario(idfuncionario));


insert into funcionario(nome, salario, dataadmissao)
values('Ana Paula', 2000, '2016-05-16');

insert into funcionario(nome, salario, dataadmissao)
values('Joao Pedro', 3000, '2016-05-13');

select * from funcionario;


insert into endereco(logradouro, cidade, estado, cep, idfuncionario)
values('Rua A, Centro', 'Sao Paulo', 'SP', '25000-000', 1);

insert into endereco(logradouro, cidade, estado, cep, idfuncionario)
values('Rua B, Centro', 'Rio de Janeiro', 'RJ', '26000-000', 2);

select * from endereco;


select
      f.idfuncionario,
      f.nome,
      f.salario,
      f.dataadmissao,
      e.idendereco,
      e.logradouro,
      e.cidade,
      e.estado,
      e.cep
from funcionario f
inner join endereco e
on f.idfuncionario = e.idfuncionario\G
```

Criando as Classes de entidade:

```
package entities;

import java.util.Date;

public class Funcionario {

	private Integer idFuncionario;
	private String nome;
	private Double salario;
	private Date dataAdmissao;

	// Relacionamento -> TEM 1 Endereco
	private Endereco endereco; //Associação

	public Funcionario() {
		// TODO Auto-generated constructor stub
	}

	public Funcionario(Integer idFuncionario, String nome,
			Double salario, Date dataAdmissao) {
		this.idFuncionario = idFuncionario;
		this.nome = nome;
		this.salario = salario;
		this.dataAdmissao = dataAdmissao;
	}

	public Funcionario(Integer idFuncionario, String nome,
			Double salario, Date dataAdmissao,
			Endereco endereco) {
		this(idFuncionario, nome, salario, dataAdmissao);
		this.endereco = endereco;
	}

	public Integer getIdFuncionario() {
		return idFuncionario;
	}

	public void setIdFuncionario(Integer idFuncionario) {
		this.idFuncionario = idFuncionario;
	}


	public String getNome() {
		return nome;
```

```
	}

	public void setNome(String nome) {
		this.nome = nome;
	}

	public Double getSalario() {
		return salario;
	}

	public void setSalario(Double salario) {
		this.salario = salario;
	}

	public Date getDataAdmissao() {
		return dataAdmissao;
	}

	public void setDataAdmissao(Date dataAdmissao) {
		this.dataAdmissao = dataAdmissao;
	}

	public Endereco getEndereco() {
		return endereco;
	}

	public void setEndereco(Endereco endereco) {
		this.endereco = endereco;
	}

	@Override
	public String toString() {
		return "Funcionario [idFuncionario="
				+ idFuncionario
				+ ", nome=" + nome
				+ ", salario=" + salario
				+ ", dataAdmissao=" + dataAdmissao + "]";
	}

}



package entities;
```

```
public class Endereco {

	private Integer idEndereco;
	private String logradouro;
	private String cidade;
	private String estado;
	private String cep;

	// Relacionamento -> TER 1
	private Funcionario funcionario; // Associação

	public Endereco() {
		// TODO Auto-generated constructor stub
	}

	public Endereco(Integer idEndereco, String logradouro,
		String cidade, String estado, String cep) {
		this.idEndereco = idEndereco;
		this.logradouro = logradouro;
		this.cidade = cidade;
		this.estado = estado;
		this.cep = cep;
	}

	public Endereco(Integer idEndereco, String logradouro,
		String cidade, String estado, String cep,
			Funcionario funcionario) {
		this(idEndereco, logradouro, cidade, estado, cep);
		this.funcionario = funcionario;
	}

	public Integer getIdEndereco() {
		return idEndereco;
	}

	public void setIdEndereco(Integer idEndereco) {
		this.idEndereco = idEndereco;
	}

	public String getLogradouro() {
		return logradouro;
	}

	public void setLogradouro(String logradouro) {
		this.logradouro = logradouro;
	}
```

```
	public String getCidade() {
		return cidade;
	}

	public void setCidade(String cidade) {
		this.cidade = cidade;
	}

	public String getEstado() {
		return estado;
	}

	public void setEstado(String estado) {
		this.estado = estado;
	}

	public String getCep() {
		return cep;
	}

	public void setCep(String cep) {
		this.cep = cep;
	}

	public Funcionario getFuncionario() {
		return funcionario;
	}

	public void setFuncionario(Funcionario funcionario) {
		this.funcionario = funcionario;
	}

	@Override
	public String toString() {
		return "Endereco [idEndereco=" + idEndereco
			+ ", logradouro=" + logradouro + ", cidade="
			+ cidade + ", estado="
			+ estado + ", cep=" + cep + "]";
	}

}
```

Criando uma Classe para tratamento da Data do Mysql em Java

package util;

```
import java.text.SimpleDateFormat;
import java.util.Date;

public class FormatacaoData {

	//método para receber um Date e retorna-lo
	//no padrão yyyy-MM-dd
	public static String convertToString(Date data){

		//classe para formatação de datas no java..
		SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
		return sdf.format(data); //retornando a data formatada..
	}

}
```

Criando Classe Dao:
Data Access Object...

```
package persistence;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class Dao {

	private static final String DRIVER = "com.mysql.jdbc.Driver";
	private static final String URL = "jdbc:mysql://localhost:3306/aula04b";
	private static final String USER = "root";
	private static final String PASSWORD = "brqbrq";

	protected Connection con;
	protected PreparedStatement stmt;
	protected CallableStatement call;
	protected ResultSet rs;

	protected void openConnection() throws Exception{
		Class.forName(DRIVER);
		con = DriverManager.getConnection(URL, USER, PASSWORD);
	}
```

```
        protected void closeConnection() throws Exception{
                if(con != null){
                        con.close();
                }
        }
}
```

Criando a classe FuncionarioDao.java

```
package persistence;

import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

import entities.Endereco;
import entities.Funcionario;
import util.FormatacaoData;

public class FuncionarioDao extends Dao{

      public Integer insert(Funcionario f) throws Exception{

            String query = "insert into Funcionario(nome, salario,
                              dataadmissao) "
                               + "values(?, ?, ?)";

            openConnection(); //abrir conexão..

            stmt = con.prepareStatement(query, Statement
                                    .RETURN_GENERATED_KEYS);
            stmt.setString(1, f.getNome());
            stmt.setDouble(2, f.getSalario());
            stmt.setString(3, FormatacaoData.convertToString
                              (f.getDataAdmissao()));
            stmt.execute();

            rs = stmt.getGeneratedKeys();
            //executar a consulta que irá obter a chave..
            rs.next();
            //mover o cursor do ResultSet para o registro da chave..

            Integer chave = rs.getInt(1); //1) unico dado..

            stmt.close();
            closeConnection(); //fechar conexão..

            return chave; //retornar a chave..
```

```
    }

    //método para listar os funcionarios da tabela com endereco.
    public List<Funcionario> findAll() throws Exception{

        String query = "select f.idfuncionario, f.nome, f.salario,
                            f.dataadmissao, "
                     + "e.idendereco, e.logradouro, e.cidade,
                            e.estado, e.cep "
                     + "from funcionario as f inner join
                            endereco as e "
                     + "on f.idfuncionario =
                            e.idfuncionario";

        openConnection();

        stmt = con.prepareStatement(query);
        rs = stmt.executeQuery();

        List<Funcionario> lista = new ArrayList<Funcionario>();
        //lista vazia..

        while(rs.next()){ //enquanto houver registros..

            Funcionario f = new Funcionario();
            f.setEndereco(new Endereco());

            f.setIdFuncionario(rs.getInt("idfuncionario"));
            f.setNome(rs.getString("nome"));
            f.setSalario(rs.getDouble("salario"));
            f.getEndereco().setIdEndereco
                  (rs.getInt("idendereco"));
            f.getEndereco().setLogradouro
                  (rs.getString("logradouro"));
            f.getEndereco().setCidade(rs.getString("cidade"));
            f.getEndereco().setEstado(rs.getString("estado"));
            f.getEndereco().setCep(rs.getString("cep"));

            lista.add(f); //adicionando na lista..
        }

        stmt.close();
        closeConnection();

        return lista; //retornando a lista..
    }
}
```

Criando a Classe EnderecoDao.java

```
package persistence;

import entities.Endereco;

public class EnderecoDao extends Dao{

	//método para gravar o endereco na base de dados..
	public void insert(Endereco e) throws Exception{

		String query = "insert into endereco(logradouro,
			cidade, estado, cep, idfuncionario) "
			 + "values(?, ?, ?, ?, ?)";

		openConnection();

		stmt = con.prepareStatement(query);
		stmt.setString(1, e.getLogradouro());
		stmt.setString(2, e.getCidade());
		stmt.setString(3, e.getEstado());
		stmt.setString(4, e.getCep());
		stmt.setInt(5, e.getFuncionario()
				.getIdFuncionario()); //foreign key..
		stmt.execute();
		stmt.close();

		closeConnection();
	}
}
```

Testando no MAIN:

```
package principal;

import java.util.Calendar;
import java.util.GregorianCalendar;

import entities.Endereco;
import entities.Funcionario;
import persistence.EnderecoDao;
import persistence.FuncionarioDao;

public class Main {

	public static void main(String[] args) {
```

```
Funcionario f = new Funcionario();
f.setNome("Sergio Mendes");
f.setSalario(1000.0);

//gerar uma data no java..
//ano, mes - 1, dia
Calendar cal = new GregorianCalendar(2016, 4, 16);
f.setDataAdmissao(cal.getTime()); //passando como date..

Endereco e = new Endereco();
e.setLogradouro("Rua Boa Vista 254, 9 andar");
e.setCidade("Sao Paulo");
e.setEstado("SP");
e.setCep("25000-000");

try{

	FuncionarioDao fd = new FuncionarioDao();
	Integer idFuncionario = fd.insert(f); //gravando..

	System.out.println("Dados gravados com sucesso.");
	System.out.println("Id do Funcionario: " + idFuncionario);

	//incluindo o id no objeto funcionario..
	f.setIdFuncionario(idFuncionario);
	e.setFuncionario(f); //relacionando o endereco ao funcionario..

	EnderecoDao ed = new EnderecoDao();
	ed.insert(e); //gravando o endereco..

	System.out.println("Endereco cadastrado com sucesso.");

	//exibir os funcionarios com endereco..
	System.out.println("\nConsulta de Funcionarios:");

	for(Funcionario func : fd.findAll()){

		System.out.println(func); //toString()
		System.out.println(func.getEndereco()); //toString()
		System.out.println("...");
	}

}
```

```
                catch(Exception ex){
                        System.out.println("Erro: " + ex.getMessage());
                }
        }
}
```

Java Database Connectivity ou **JDBC** é um conjunto de classes e interfaces (API) escritas em Java que fazem o envio de instruções SQL para qualquer banco de dadosrelacional; Api de baixo nível e base para api's de alto nível; Amplia o que você pode fazer com Java; Possibilita o uso de bancos de dados já instalados;