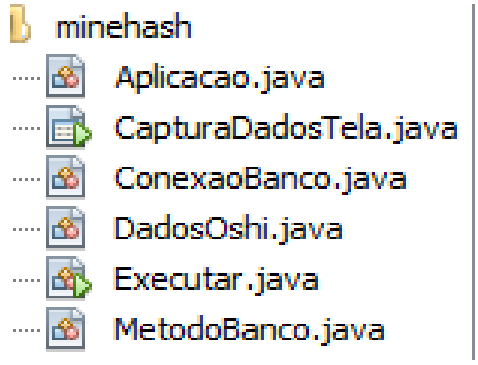
RESULTADO PARA ESTE O CÓDIGO NO JAVA:

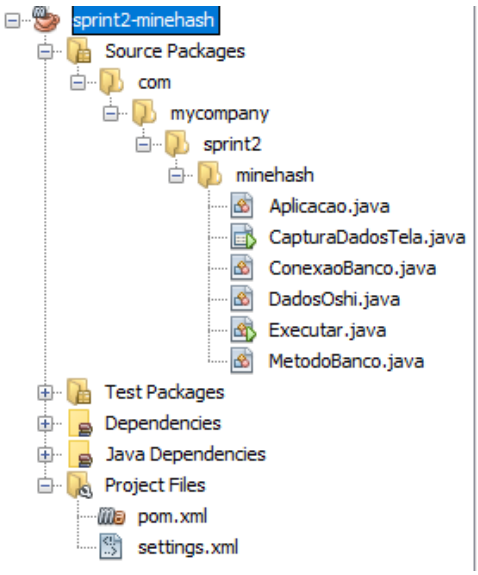


PASSO-A-PASSO PROJETO NOVO

1º no Java crie um projeto novo

2º POM-XML

Neste projeto novo, vá na pasta Project Files => pom.xml



Repare no corpo do texto:

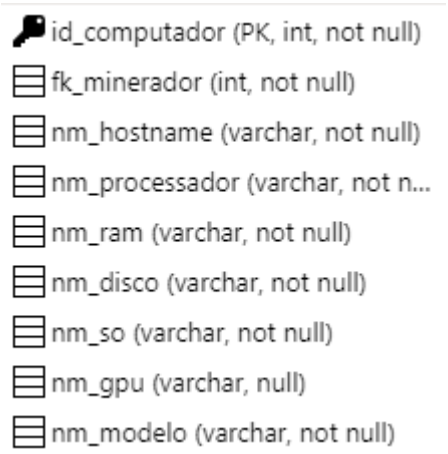
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.mycompany</groupId>
    <artifactId>sprint2-minehash</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>
    <dependencies>
        <dependency>
```

Há a tag dependencies, dentro da mesma cole o texto a seguir, que são todas as dependências que serão utilizadas no projeto:

```
<dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-jdbc</artifactId>
            <version>5.1.4.RELEASE</version>
        </dependency>

        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-dbcp2</artifactId>
            <version>2.5.0</version>
        </dependency>
 <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.17</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/com.microsoft.sqlserver/mssql-jdbc -->
        <dependency>
            <groupId>com.microsoft.sqlserver</groupId>
            <artifactId>mssql-jdbc</artifactId>
            <version>6.4.0.jre8</version>
            <type>jar</type>
        </dependency>

        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>RELEASE</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/com.github.oshi/oshi-core -->
        <dependency>
            <groupId>com.github.oshi</groupId>
            <artifactId>oshi-core</artifactId>
            <version>3.5.0</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/com.github.oshi/oshi-core -->
        <dependency>
            <groupId>com.github.oshi</groupId>
            <artifactId>oshi-core</artifactId>
            <version>3.13.3</version>
        </dependency>
    </dependencies>
```

NO AZURE

No Azure realize o seguinte código SQLServer, faça um select para confirmar a criação da tabela e popule-a com alguns dados :



JFRAME

Crie um JFrame no java conforme abaixo:
Atenção : repare que os campos vêm de encontro com a tabela criada no SQL e que onde se apresentarão os dados, são label's

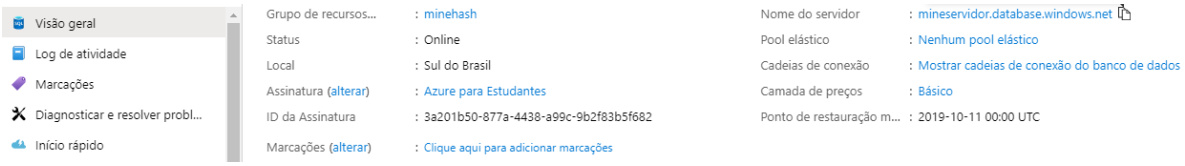
CLASSE CONEXAOBANCO

1º Criar uma Java Class chamada: ConexaoBanco
2º Criar objeto conexao com a classe BasicDataSource
3º Crie um construtor deste e dentro alterar o texto para o uso do exception TRY/CATH:

```
try {
        conexao.setDriverClassName("com.microsoft.sqlserver.jdbc.SQLServerDriver");

        conexao.setUrl("jdbc:sqlserver://mineservidor.database.windows.net:1433;"
                +
"database=tb_computador;user=minehash@mineservidor;password=Bandtec@2019;"
                + "encrypt=true;trustServerCertificate=false;"
                + "hostNameInCertificate=*.database.windows.net;loginTimeout=30;");


    } catch (Exception e) {
        e.printStackTrace();
    }
```

ATENÇÃO!
Alterar os campos conforme o seu projeto:

jdbc:sqlserver:// e colocar na frente o código que pegará no azure



database=equivale ao nome da tabela que receberá os dados, aqui no exemplo se chama: tb_computador
user=minehash@mineservidor;password=Bandtec@2019
Neste campo o user representa 2 dados:
- nome da tabela @ nome do servidor, o qual vc pega no exemplo acima;

password=é a senha deste banco

4ºRepare que o erro esta sendo tratado com try/catch

Dentro de Try coloca todo o código, bloco, que é monitorado para erros;
Em catch, colocar o tipo da exceção e nome do objeto, caso não tenha, se trata de modo genérico (Exception e) nos projetos reais.

5º Criar o construtor template e instanciar para o objeto conexão:

```
public JdbcTemplate template() {
    JdbcTemplate jdbcTemplate = new JdbcTemplate(conexao);
    return jdbcTemplate;
}
```

SUPERCLASSE DADOSOSHI

Nesta classe serão criados os atributos a serem “capturados” pela oshi.

1º Criar a classe DadosOshi
Abaixo de package, conforme mostrado na imagem, fazer o import de:

import oshi.SystemInfo;
import oshi.hardware.*;
import oshi.software.os.*;
import oshi.util.FormatUtil;

```
package com.mycompany.sprint2.minehash;

import oshi.SystemInfo;
import oshi.hardware.*;
import oshi.software.os.*;
import oshi.util.FormatUtil;

public class DadosOshi {
```

2ºDentro da classe oshi,criar os construtores e instanciar:

```
SystemInfo sistema = new SystemInfo();
HardwareAbstractionLayer hardware = sistema.getHardware();
OperatingSystem sistemaOperacional = sistema.getOperatingSystem();
```

3ºAbaixo, colocar todos os atributos a serem capturados, com os seus respectivos tipos. O texto completo, ficará conforme a imagem abaixo.

```
public class DadosOshi {

    SystemInfo sistema = new SystemInfo();
    HardwareAbstractionLayer hardware = sistema.getHardware();
    OperatingSystem sistemaOperacional = sistema.getOperatingSystem();

    String processador;
    String memoriaTotal;
    String fabricante;
```

4º Para todos os atributos encapsular no método get
5º- colocar um construtor em cima do método getter e dentro “chamar” todas as configurações dos atributos, conforme este exemplo:

```
public DadosOshi() {
    this.hostname = sistema.getOperatingSystem().getFileSystem().toString();
    this.processador = hardware.getProcessor().getName();
    this.memoriaTotal = FormatUtil.formatBytes(hardware.getMemory().getTotal());
    this.disco = FormatUtil.formatBytes(hardware.getDiskStores().length);
    this.so = sistema.getOperatingSystem().getVersion().toString();
    this.gpu = hardware.getDisplays().toString();
    this.modelo = hardware.getComputerSystem().getManufacturer();
}
```

CLASSE METODOBANCO

Esta classe é responsável por enviar os dados para a tabela do banco de dados.

1º Aplicar o método de herança com relação a superclasse, logo, fazer o extends DadosOshi

```
public class MetodoBanco extends DadosOshi {
```

2º criar um construtor com instanciamento acerca da classe ConexaoBanco

ConexaoBanco conexao = new ConexaoBanco();

3ºCriar um método void enviarBanco, recorde que este método não retorna nada, logo, vc colocará os parâmetros que serão “puxados” da sua máquina.

4ºDentro do void, colocar:

conexao.getConnection();
conexao.template().update(

5º Dentro de update vc colocará o nome das colunas, a sequencia deve seguir o padrão da tabela feita no SQLSERVER, no meu exemplo é: hostname, processador, ram, disco,so,gpu,modelo, e na frente colocar dentro de values um total de ? igual ao número de colunas, ao final, preencher com todos os “gets” referentes aos dados que serão captados.

O resultado final será desta forma:

```
public class MetodoBanco extends DadosOshi {


    ConexaoBanco conexao = new ConexaoBanco();

    public void enviarBanco() {
        conexao.getConnection();
        conexao.template().update(
                "insert into tb_computador (hostname, processador, ram,"
                        +"disco,so,gpu,modelo) values (?,?,?,?,?,?,?)",
                getHostname(),getProcessador(),getMemoriaTotal(),getDisco(),
                getSo(),getGpu(),getModelo());
    }
}
```

CLASSE APLICACAO

1ºCriar uma nova classe chamada aplicação

2º Dentro criar 2 construtores e instanciar, conforme este exemplo:

ConexaoBanco banco = new ConexaoBanco(); =>Aqui vc estará instanciando novamente a classe ConexaoBanco, porém com um novo nome de objeto
MetodoBanco inserir = new MetodoBanco(); =>Aqui vc estará instanciando a classe MetodoBanco

3ºCriar o construtor public Aplicacao(){}

4ºDentro de aplicação, colocar:
banco.getConnection();
inserir.enviarBanco(); =>é o void criado na classe MetodoBanco

5º ao final fazer um “sout”:
System.out.println(banco.template().queryForList("select * from tb_computador"));

ATENÇÃO: alterar no campo select * from com o nome da sua tabela à frente, neste exemplo é tb_computador.

Ao final espera-se este resultado:

```
public class Aplicacao {

    ConexaoBanco banco = new ConexaoBanco(); //Aqui vc es
    MetodoBanco inserir = new MetodoBanco();

    public Aplicacao() {

        banco.getConnection();
        inserir.enviarBanco(); // é o void criado na clas

        System.out.println(banco.template().queryForList(
                sql:"select * from tb_computador"));

    }

}
```

CLASSE EXECUTAR

Como o nome diz, é responsável por executar o processo

1ºCriar uma nova classe chamada executar
2ºCriar um método main
3ºDentro do main, criar um construtor e instanciar, o nome do objeto será app
Aplicacao app = new Aplicacao(); => instanciado da classe aplicacao

Ao final espera-se este resultado:

```
public class Executar {

    public static void main(String[] args) {

        Aplicacao app = new Aplicacao();

    }
}
```