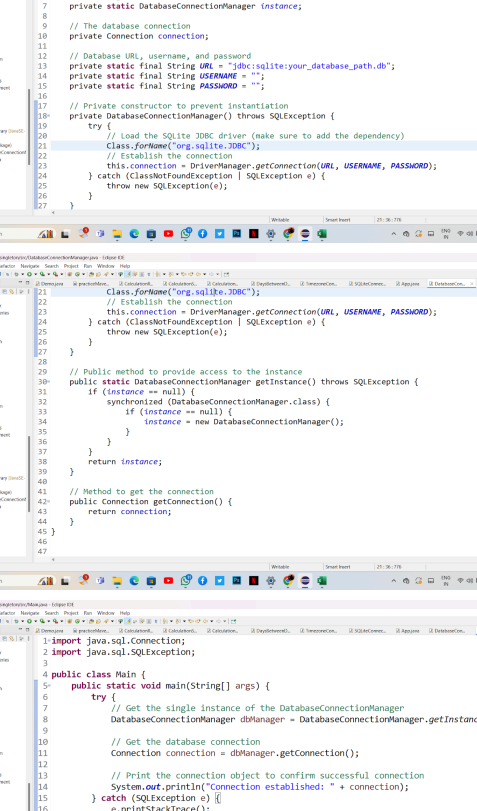


Implement a Singleton class that manages database connections. Ensure the class adheres strictly to the singleton pattern principles.

1. **Private Constructor:** The constructor is private to prevent any other class from instantiating it.
2. **Static Instance:** A static instance of the `DatabaseConnectionManager` is declared. This is the single instance that will be used throughout the application.
3. **getInstance() Method:** This method provides a global point of access to the instance. It ensures that only one instance is created using double-checked locking to maintain thread safety.
4. **getConnection() Method:** This method returns the database connection object.
5. **Singleton Pattern:** Ensures only one instance of `DatabaseConnectionManager` is created.
6. **Database Connection Management:** Manages and provides a single point of access to the database connection.
7. **Thread Safety:** Uses double-checked locking to ensure thread safety during instance creation.

By following this structure, you ensure that your application has a single, globally accessible instance for managing database connections.



The image displays two screenshots of a Windows IDE (likely Visual Studio Code) showing the implementation of a JDBC connection manager in Java. The top screenshot shows the initial code structure, and the bottom screenshot shows the completed code with a main method for testing.

Top Screenshot: DatabaseConnectionManager.java

```

1 package jdbc;
2 import java.sql.*;
3 import java.sql.SQLException;
4
5 public class DatabaseConnectionManager {
6     // The single instance of the class
7     private static DatabaseConnectionManager instance;
8
9     // The database connection
10    private Connection connection;
11
12    // Database URL, username, and password
13    private static final String URL = "jdbc:sqlite:your_database_path.db";
14    private static final String USERNAME = "";
15    private static final String PASSWORD = "";
16
17    // Private constructor to prevent instantiation
18    private DatabaseConnectionManager() throws SQLException {
19        try {
20            // Load the SQLite JDBC driver (make sure to add the dependency)
21            Class.forName("org.sqlite.JDBC");
22        } catch (ClassNotFoundException | SQLException e) {
23            // Establish the connection
24            this.connection = DriverManager.getConnection(URL, USERNAME, PASSWORD);
25        } catch (ClassNotFoundException | SQLException e) {
26            throw new SQLException(e);
27        }
28    }
29
30    // Public method to provide access to the instance
31    public static DatabaseConnectionManager getInstance() throws SQLException {
32        if (instance == null) {
33            synchronized (DatabaseConnectionManager.class) {
34                if (instance == null) {
35                    instance = new DatabaseConnectionManager();
36                }
37            }
38        }
39        return instance;
40    }
41
42    // Method to get the connection
43    public Connection getConnection() {
44        return connection;
45    }
46
47    // Setter method
48    public void setConnection(Connection connection) {
49        this.connection = connection;
50    }
51
52    // Getter method
53    public Connection getConnection() {
54        return this.connection;
55    }
56
57    // Close the connection
58    public void closeConnection() throws SQLException {
59        if (connection != null) {
60            connection.close();
61        }
62    }
63
64    // Main method for testing
65    public static void main(String[] args) {
66        try {
67            DatabaseConnectionManager dbManager = DatabaseConnectionManager.getInstance();
68            Connection connection = dbManager.getConnection();
69
70            // Print the connection URL to confirm successful connection
71            System.out.println("Connection established: " + connection);
72
73            // Close the connection
74            dbManager.closeConnection();
75        } catch (SQLException e) {
76            e.printStackTrace();
77        }
78    }
79
80    // Close the connection
81    public void closeConnection() throws SQLException {
82        if (connection != null) {
83            connection.close();
84        }
85    }
86
87    // Main method for testing
88    public static void main(String[] args) {
89        try {
90            DatabaseConnectionManager dbManager = DatabaseConnectionManager.getInstance();
91            Connection connection = dbManager.getConnection();
92
93            // Print the connection URL to confirm successful connection
94            System.out.println("Connection established: " + connection);
95
96            // Close the connection
97            dbManager.closeConnection();
98        } catch (SQLException e) {
99            e.printStackTrace();
100        }
101    }
102
103    // Close the connection
104    public void closeConnection() throws SQLException {
105        if (connection != null) {
106            connection.close();
107        }
108    }
109
110    // Main method for testing
111    public static void main(String[] args) {
112        try {
113            DatabaseConnectionManager dbManager = DatabaseConnectionManager.getInstance();
114            Connection connection = dbManager.getConnection();
115
116            // Print the connection URL to confirm successful connection
117            System.out.println("Connection established: " + connection);
118
119            // Close the connection
120            dbManager.closeConnection();
121        } catch (SQLException e) {
122            e.printStackTrace();
123        }
124    }
125
126    // Close the connection
127    public void closeConnection() throws SQLException {
128        if (connection != null) {
129            connection.close();
130        }
131    }
132
133    // Main method for testing
134    public static void main(String[] args) {
135        try {
136            DatabaseConnectionManager dbManager = DatabaseConnectionManager.getInstance();
137            Connection connection = dbManager.getConnection();
138
139            // Print the connection URL to confirm successful connection
140            System.out.println("Connection established: " + connection);
141
142            // Close the connection
143            dbManager.closeConnection();
144        } catch (SQLException e) {
145            e.printStackTrace();
146        }
147    }
148
149    // Close the connection
150    public void closeConnection() throws SQLException {
151        if (connection != null) {
152            connection.close();
153        }
154    }
155
156    // Main method for testing
157    public static void main(String[] args) {
158        try {
159            DatabaseConnectionManager dbManager = DatabaseConnectionManager.getInstance();
160            Connection connection = dbManager.getConnection();
161
162            // Print the connection URL to confirm successful connection
163            System.out.println("Connection established: " + connection);
164
165            // Close the connection
166            dbManager.closeConnection();
167        } catch (SQLException e) {
168            e.printStackTrace();
169        }
170    }
171
172    // Close the connection
173    public void closeConnection() throws SQLException {
174        if (connection != null) {
175            connection.close();
176        }
177    }
178
179    // Main method for testing
180    public static void main(String[] args) {
181        try {
182            DatabaseConnectionManager dbManager = DatabaseConnectionManager.getInstance();
183            Connection connection = dbManager.getConnection();
184
185            // Print the connection URL to confirm successful connection
186            System.out.println("Connection established: " + connection);
187
188            // Close the connection
189            dbManager.closeConnection();
190        } catch (SQLException e) {
191            e.printStackTrace();
192        }
193    }
194
195    // Close the connection
196    public void closeConnection() throws SQLException {
197        if (connection != null) {
198            connection.close();
199        }
200    }
201
202    // Main method for testing
203    public static void main(String[] args) {
204        try {
205            DatabaseConnectionManager dbManager = DatabaseConnectionManager.getInstance();
206            Connection connection = dbManager.getConnection();
207
208            // Print the connection URL to confirm successful connection
209            System.out.println("Connection established: " + connection);
210
211            // Close the connection
212            dbManager.closeConnection();
213        } catch (SQLException e) {
214            e.printStackTrace();
215        }
216    }
217
218    // Close the connection
219    public void closeConnection() throws SQLException {
220        if (connection != null) {
221            connection.close();
222        }
223    }
224
225    // Main method for testing
226    public static void main(String[] args) {
227        try {
228            DatabaseConnectionManager dbManager = DatabaseConnectionManager.getInstance();
229            Connection connection = dbManager.getConnection();
230
231            // Print the connection URL to confirm successful connection
232            System.out.println("Connection established: " + connection);
233
234            // Close the connection
235            dbManager.closeConnection();
236        } catch (SQLException e) {
237            e.printStackTrace();
238        }
239    }
240
241    // Close the connection
242    public void closeConnection() throws SQLException {
243        if (connection != null) {
244            connection.close();
245        }
246    }
247
248    // Main method for testing
249    public static void main(String[] args) {
250        try {
251            DatabaseConnectionManager dbManager = DatabaseConnectionManager.getInstance();
252            Connection connection = dbManager.getConnection();
253
254            // Print the connection URL to confirm successful connection
255            System.out.println("Connection established: " + connection);
256
257            // Close the connection
258            dbManager.closeConnection();
259        } catch (SQLException e) {
260            e.printStackTrace();
261        }
262    }
263
264    // Close the connection
265    public void closeConnection() throws SQLException {
266        if (connection != null) {
267            connection.close();
268        }
269    }
270
271    // Main method for testing
272    public static void main(String[] args) {
273        try {
274            DatabaseConnectionManager dbManager = DatabaseConnectionManager.getInstance();
275            Connection connection = dbManager.getConnection();
276
277            // Print the connection URL to confirm successful connection
278            System.out.println("Connection established: " + connection);
279
280            // Close the connection
281            dbManager.closeConnection();
282        } catch (SQLException e) {
283            e.printStackTrace();
284        }
285    }
286
287    // Close the connection
288    public void closeConnection() throws SQLException {
289        if (connection != null) {
290            connection.close();
291        }
292    }
293
294    // Main method for testing
295    public static void main(String[] args) {
296        try {
297            DatabaseConnectionManager dbManager = DatabaseConnectionManager.getInstance();
298            Connection connection = dbManager.getConnection();
299
300            // Print the connection URL to confirm successful connection
301            System.out.println("Connection established: " + connection);
302
303            // Close the connection
304            dbManager.closeConnection();
305        } catch (SQLException e) {
306            e.printStackTrace();
307        }
308    }
309
310    // Close the connection
311    public void closeConnection() throws SQLException {
312        if (connection != null) {
313            connection.close();
314        }
315    }
316
317    // Main method for testing
318    public static void main(String[] args) {
319        try {
320            DatabaseConnectionManager dbManager = DatabaseConnectionManager.getInstance();
321            Connection connection = dbManager.getConnection();
322
323            // Print the connection URL to confirm successful connection
324            System.out.println("Connection established: " + connection);
325
326            // Close the connection
327            dbManager.closeConnection();
328        } catch (SQLException e) {
329            e.printStackTrace();
330        }
331    }
332
333    // Close the connection
334    public void closeConnection() throws SQLException {
335        if (connection != null) {
336            connection.close();
337        }
338    }
339
340    // Main method for testing
341    public static void main(String[] args) {
342        try {
343            DatabaseConnectionManager dbManager = DatabaseConnectionManager.getInstance();
344            Connection connection = dbManager.getConnection();
345
346            // Print the connection URL to confirm successful connection
347            System.out.println("Connection established: " + connection);
348
349            // Close the connection
350            dbManager.closeConnection();
351        } catch (SQLException e) {
352            e.printStackTrace();
353        }
354    }
355
356    // Close the connection
357    public void closeConnection() throws SQLException {
358        if (connection != null) {
359            connection.close();
360        }
361    }
362
363    // Main method for testing
364    public static void main(String[] args) {
365        try {
366            DatabaseConnectionManager dbManager = DatabaseConnectionManager.getInstance();
367            Connection connection = dbManager.getConnection();
368
369            // Print the connection URL to confirm successful connection
370            System.out.println("Connection established: " + connection);
371
372            // Close the connection
373            dbManager.closeConnection();
374        } catch (SQLException e) {
375            e.printStackTrace();
376        }
377    }
378
379    // Close the connection
380    public void closeConnection() throws SQLException {
381        if (connection != null) {
382            connection.close();
383        }
384    }
385
386    // Main method for testing
387    public static void main(String[] args) {
388        try {
389            DatabaseConnectionManager dbManager = DatabaseConnectionManager.getInstance();
390            Connection connection = dbManager.getConnection();
391
392            // Print the connection URL to confirm successful connection
393            System.out.println("Connection established: " + connection);
394
395            // Close the connection
396            dbManager.closeConnection();
397        } catch (SQLException e) {
398            e.printStackTrace();
399        }
400    }
401
402    // Close the connection
403    public void closeConnection() throws SQLException {
404        if (connection != null) {
405            connection.close();
406        }
407    }
408
409    // Main method for testing
410    public static void main(String[] args) {
411        try {
412            DatabaseConnectionManager dbManager = DatabaseConnectionManager.getInstance();
413            Connection connection = dbManager.getConnection();
414
415            // Print the connection URL to confirm successful connection
416            System.out.println("Connection established: " + connection);
417
418            // Close the connection
419            dbManager.closeConnection();
420        } catch (SQLException e) {
421            e.printStackTrace();
422        }
423    }
424
425    // Close the connection
426    public void closeConnection() throws SQLException {
427        if (connection != null) {
428            connection.close();
429        }
430    }
431
432    // Main method for testing
433    public static void main(String[] args) {
434        try {
435            DatabaseConnectionManager dbManager = DatabaseConnectionManager.getInstance();
436            Connection connection = dbManager.getConnection();
437
438            // Print the connection URL to confirm successful connection
439            System.out.println("Connection established: " + connection);
440
441            // Close the connection
442            dbManager.closeConnection();
443        } catch (SQLException e) {
444            e.printStackTrace();
445        }
446    }
447
448    // Close the connection
```

Task 2: Factory Method

Create a ShapeFactory class that encapsulates the object creation logic of different Shape objects like Circle, Square, and Rectangle.

Explanation

1. Shape Interface:

- The `Shape` interface defines a `draw` method that all shape classes must implement.

2. Concrete Classes:

- `Circle`, `Square`, and `Rectangle` classes each implement the `Shape` interface and provide their own implementation of the `draw` method.

3. ShapeFactory Class:

- The `ShapeFactory` class contains a method `getShape` that returns an instance of a specific shape based on the input string.

4. Main Class:

- The `Main` class demonstrates how to use the `ShapeFactory` to create different shapes and call their `draw` methods.

Directory Structure

src

```
├── factory
│   └── ShapeFactory.java
├── main
│   └── Main.java
└── shapes
    ├── Circle.java
    ├── Rectangle.java
    ├── Shape.java
    └── Square.java
```

```
1 // ShapeFactory.java
2 package main;
3
4 public class ShapeFactory {
5     // Method to get a Shape object based on the shape type
6     public Shape getShape(String shapeType) {
7         if (shapeType == null) {
8             return null;
9         }
10        if (shapeType.equalsIgnoreCase("CIRCLE")) {
11            return new Shape();
12        } else if (shapeType.equalsIgnoreCase("SQUARE")) {
13            return new Shape();
14        } else if (shapeType.equalsIgnoreCase("RECTANGLE")) {
15            return new Shape();
16        }
17        return null;
18    }
19 }
20
21
22
23
```

```
1 // Main.java
2 package main;
3
4 import factory.ShapeFactory;
5
6 public class Main {
7     public static void main(String[] args) {
8         ShapeFactory shapeFactory = new ShapeFactory();
9
10        // Get an object of Circle and call its draw method
11        Shape shape1 = shapeFactory.getShape("CIRCLE");
12        shape1.draw();
13
14        // Get an object of Square and call its draw method
15        Shape shape2 = shapeFactory.getShape("SQUARE");
16        shape2.draw();
17
18        // Get an object of Rectangle and call its draw method
19        Shape shape3 = shapeFactory.getShape("RECTANGLE");
20        shape3.draw();
21    }
22 }
23
```

```
1 // Circle.java
2 package shapes;
3
4 import java.awt.Shape;
5
6 public abstract class Circle implements Shape {
7     public void draw() {
8         System.out.println("Drawing a Circle");
9     }
10 }
11
```

```
1 // Square.java
2 package shapes;
3
4 import java.awt.Shape;
5
6 public abstract class Square implements Shape {
7     public void draw() {
8         System.out.println("Drawing a Square");
9     }
10 }
11
```

```
1 // Rectangle.java
2 package shapes;
3
4 import java.awt.Shape;
5
6 public abstract class Rectangle implements Shape {
7     public void draw() {
8         System.out.println("Drawing a Rectangle");
9     }
10 }
11
```

Task 3: Proxy

Create a proxy class for accessing a sensitive object that contains a secret key. The proxy should only allow access to the secret key if a correct password is provided.

Explanation

1. SensitiveObject Class:

- Holds the secret key.
- Provides a method to get the secret key.

2. SensitiveObjectProxy Class:

- Encapsulates the `SensitiveObject` and controls access to it.
- Checks the provided password against the stored password and grants or denies access to the secret key accordingly.

3. Main Class:

- Demonstrates how to use the proxy to access the secret key with correct and incorrect passwords.

Directory Structure

src

```
├── main
│   └── Main.java
└── security
    ├── SensitiveObject.java
    └── SensitiveObjectProxy.java
```

```
1 // Main.java
2
3
4 import security.SensitiveObjectProxy;
5
6 public class Main {
7     public static void main(String[] args) {
8         // Create a proxy with the secret key and the password
9         SensitiveObjectProxy proxy = new SensitiveObjectProxy("mySecretKey123", "password123");
10
11         // Try to access the secret key with the correct password
12         String correctPassword = "password123";
13         System.out.println("Attempt with correct password: " + proxy.getSecretKey(correctPassword));
14
15         // Try to access the secret key with an incorrect password
16         String incorrectPassword = "wrongPassword";
17         System.out.println("Attempt with incorrect password: " + proxy.getSecretKey(incorrectPassword));
18     }
19 }
20
```

```
1 // SensitiveObjectProxy.java
2 package security;
3
4 public class SensitiveObjectProxy {
5     private SensitiveObject sensitiveObject;
6     private String password;
7
8     public SensitiveObjectProxy(String secretKey, String password) {
9         this.sensitiveObject = new SensitiveObject(secretKey);
10        this.password = password;
11    }
12
13    public String getSecretKey(String password) {
14        if (this.password.equals(password)) {
15            return sensitiveObject.getSecretKey();
16        } else {
17            return "Access Denied: Incorrect Password";
18        }
19    }
20 }
21
```

```
1 // SensitiveObject.java
2 package security;
3
4 public class SensitiveObject {
5     private String secretKey;
6
7     public SensitiveObject(String secretKey) {
8         this.secretKey = secretKey;
9     }
10
11    public String getSecretKey() {
12        return secretKey;
13    }
14 }
15
```

```
1 // Main.java
2
3
4 import security.SensitiveObjectProxy;
5
6 public class Main {
7     public static void main(String[] args) {
8         // Create a proxy with the secret key and the password
9         SensitiveObjectProxy proxy = new SensitiveObjectProxy("mySecretKey123", "password123");
10
11         // Try to access the secret key with the correct password
12         String correctPassword = "password123";
13         System.out.println("Attempt with correct password: " + proxy.getSecretKey(correctPassword));
14
15         // Try to access the secret key with an incorrect password
16         String incorrectPassword = "wrongPassword";
17         System.out.println("Attempt with incorrect password: " + proxy.getSecretKey(incorrectPassword));
18     }
19 }
20
```

```
Attempt with correct password: mySecretKey123
Attempt with incorrect password: Access Denied: Incorrect Password
```

Task 4: Strategy

Develop a Context class that can use different SortingStrategy algorithms interchangeably to sort a collection of numbers

Explanation of the code:

- `SortingStrategy` is an interface that defines the `sort` method.
- `BubbleSortStrategy` and `QuickSortStrategy` are classes that implement the `SortingStrategy` interface with their respective sorting algorithms.
- `Context` class manages the sorting strategy and provides a method to sort numbers using the selected strategy.
- In the `Main` class, we create an instance of `Context` and demonstrate sorting numbers using both Bubble Sort and Quick Sort strategies interchangeably.

The execution flow:

1. We create a `Context` object.
2. We initialize an array of numbers to sort.
3. We set the sorting strategy to Bubble Sort, sort the numbers, and print the sorted array.
4. We set the sorting strategy to Quick Sort, sort the numbers again, and print the sorted array.

When you run the `Main` class, you'll see the sorted arrays using both Bubble Sort and Quick Sort strategies.

```
// SortingStrategy.java
public interface SortingStrategy {
    void sort(int[] numbers);
}

// BubbleSortStrategy.java
public class BubbleSortStrategy implements SortingStrategy {
    @Override
    public void sort(int[] numbers) {
        int n = numbers.length;
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (numbers[j] > numbers[j + 1]) {
                    // Swap
                    int temp = numbers[j];
                    numbers[j] = numbers[j + 1];
                    numbers[j + 1] = temp;
                }
            }
        }
    }
}

// QuickSortStrategy.java
public class QuickSortStrategy implements SortingStrategy {
    @Override
    public void sort(int[] numbers) {
        quickSort(numbers, 0, numbers.length - 1);
    }

    private void quickSort(int[] arr, int low, int high) {
        if (low < high) {
            int pi = partition(arr, low, high);
            quickSort(arr, low, pi - 1);
            quickSort(arr, pi + 1, high);
        }
    }

    private int partition(int[] arr, int low, int high) {
        int pivot = arr[high];
        int i = low - 1;
        for (int j = low; j < high; j++) {
            if (arr[j] < pivot) {
                i++;
                // Swap
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        // Swap
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;
        return i + 1;
    }
}

// Context.java
public class Context {
    private SortingStrategy sortingStrategy;

    public Context() {
        this.sortingStrategy = new BubbleSortStrategy();
    }

    public void setSortingStrategy(SortingStrategy sortingStrategy) {
        this.sortingStrategy = sortingStrategy;
    }

    public void sortNumbers(int[] numbers) {
        if (sortingStrategy != null) {
            sortingStrategy.sort(numbers);
        } else {
            System.out.println("Please set a sorting strategy.");
        }
    }
}

// Main.java
import java.util.Arrays;

public class Main {
    public static void main(String[] args) {
        Context context = new Context();

        int[] numbersToSort = {5, 1, 4, 2, 8};
        SortingStrategy bubbleSortStrategy = new BubbleSortStrategy();
        context.setSortingStrategy(bubbleSortStrategy);
        context.sortNumbers(numbersToSort);
        System.out.println("Sorted using Bubble Sort: " + Arrays.toString(numbersToSort));

        SortingStrategy quickSortStrategy = new QuickSortStrategy();
        context.setSortingStrategy(quickSortStrategy);
        context.sortNumbers(numbersToSort);
        System.out.println("Sorted using Quick Sort: " + Arrays.toString(numbersToSort));
    }
}
```