Day 23:

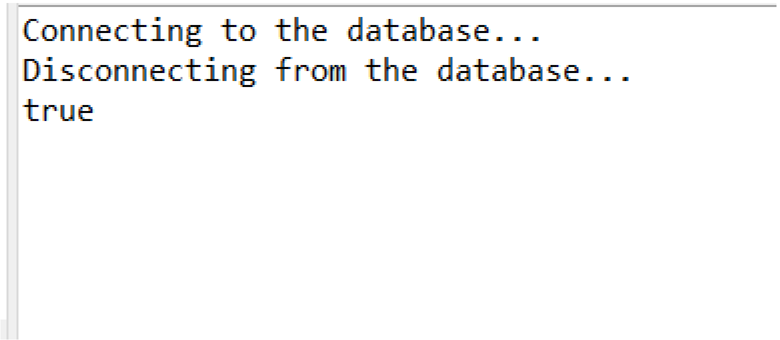
Task 1: Singleton

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

Solution:

```java
public class DatabaseConnectionManager {
    // Step 2: Create a private static instance of the class
    private static DatabaseConnectionManager instance;

    // Step 3: Create a private constructor to prevent instantiation
    private DatabaseConnectionManager() {
        // Initialize the database connection here
    }

    // Step 4: Provide a public static method to get the instance
    public static synchronized DatabaseConnectionManager getInstance() {
        if (instance == null) {
            instance = new DatabaseConnectionManager();
        }
        return instance;
    }

    // Method to connect to the database
    public void connect() {
        // Database connection code here
        System.out.println("Connecting to the database...");
    }

    // Method to disconnect from the database
    public void disconnect() {
        // Database disconnection code here
        System.out.println("Disconnecting from the database...");
    }
}


package Task;

public class Main {
    public static void main(String[] args) {
        // Get the instance of the Singleton class
        DatabaseConnectionManager dbManager = 
DatabaseConnectionManager.getInstance();

        // Use the instance to connect to the database
        dbManager.connect();

        // Use the instance to disconnect from the database
        dbManager.disconnect();

        // Verify that the same instance is used
        DatabaseConnectionManager dbManager2 = 
DatabaseConnectionManager.getInstance();
        System.out.println(dbManager == dbManager2); // Should print "true"
    }
}
```

Output:



```
Connecting to the database...
Disconnecting from the database...
true
```

Task 2: Factory Method

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

```
package Task;

//Shape interface
interface Shape {
 void draw();
}

//Circle class implementing Shape interface
class Circle implements Shape {
 @Override
 public void draw() {
     System.out.println("Drawing a circle");
 }
}

//Square class implementing Shape interface
class Square implements Shape {
 @Override
 public void draw() {
     System.out.println("Drawing a square");
 }
}
```

```java
//Rectangle class implementing Shape interface
class Rectangle implements Shape {
 @Override
 public void draw() {
     System.out.println("Drawing a rectangle");
 }
}

//ShapeFactory interface
interface ShapeFactory {
 Shape createShape();
}

//CircleFactory class implementing ShapeFactory interface
class CircleFactory implements ShapeFactory {
 @Override
 public Shape createShape() {
     return new Circle();
 }
}

//SquareFactory class implementing ShapeFactory interface
class SquareFactory implements ShapeFactory {
 @Override
 public Shape createShape() {
     return new Square();
 }
}

//RectangleFactory class implementing ShapeFactory interface
class RectangleFactory implements ShapeFactory {
 @Override
 public Shape createShape() {
     return new Rectangle();
 }
}

//Main class to test the ShapeFactory
public class Main {
 public static void main(String[] args) {
     ShapeFactory circleFactory = new CircleFactory();
     ShapeFactory squareFactory = new SquareFactory();
     ShapeFactory rectangleFactory = new RectangleFactory();

     Shape circle = circleFactory.createShape();
     Shape square = squareFactory.createShape();
     Shape rectangle = rectangleFactory.createShape();

     circle.draw();
     square.draw();
     rectangle.draw();
 }
}
```

Task 3: Proxy

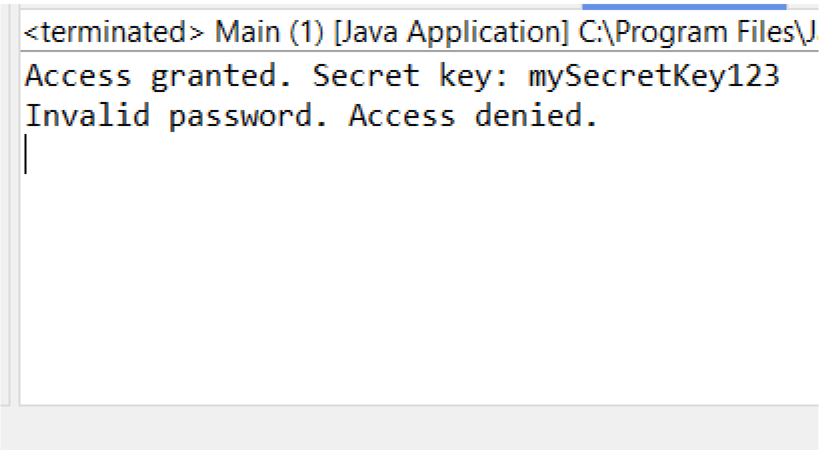
Create a proxy class for accessing a sensitive object that contains a secret key. The proxy should only allow access to the secret key if a correct password is provided.

Solution:

```java
package Task;

public class SensitiveObject {
    private String secretKey;

    public SensitiveObject(String secretKey) {
        this.secretKey = secretKey;
    }

    public String getSecretKey() {
        return secretKey;
    }
}
```

```java
package Task;

public class SensitiveObjectProxy {
    private SensitiveObject sensitiveObject;
    private String password;

    public SensitiveObjectProxy(String secretKey, String password) {
        this.sensitiveObject = new SensitiveObject(secretKey);
        this.password = password;
    }

    public String getSecretKey(String inputPassword) {
        if (this.password.equals(inputPassword)) {
            return sensitiveObject.getSecretKey();
        } else {
            throw new SecurityException("Invalid password. Access denied.");
        }
    }
}
```

```java
package Task;
public class SensitiveObjectProxy {
    private SensitiveObject sensitiveObject;
    private String password;

    public SensitiveObjectProxy(String secretKey, String password) {
        this.sensitiveObject = new SensitiveObject(secretKey);
        this.password = password;
    }

    public String getSecretKey(String inputPassword) {
        if (this.password.equals(inputPassword)) {
            return sensitiveObject.getSecretKey();
        } else {
            throw new SecurityException("Invalid password. Access denied.");
        }
    }
}

package Task;
public class Main {
    public static void main(String[] args) {
        // Initialize the proxy with the secret key and password
        SensitiveObjectProxy proxy = new SensitiveObjectProxy("mySecretKey123",
"password123");

        // Attempt to access the secret key with the correct password
        try {
            String key = proxy.getSecretKey("password123");
            System.out.println("Access granted. Secret key: " + key);
        } catch (SecurityException e) {
            System.out.println(e.getMessage());
        }

        // Attempt to access the secret key with an incorrect password
        try {
            String key = proxy.getSecretKey("wrongPassword");
            System.out.println("Access granted. Secret key: " + key);
        } catch (SecurityException e) {
            System.out.println(e.getMessage());
        }
    }
}
```



```
<terminated> Main (1) [Java Application] C:\Program Files\J
Access granted. Secret key: mySecretKey123
Invalid password. Access denied.
```

Task 4: Strategy

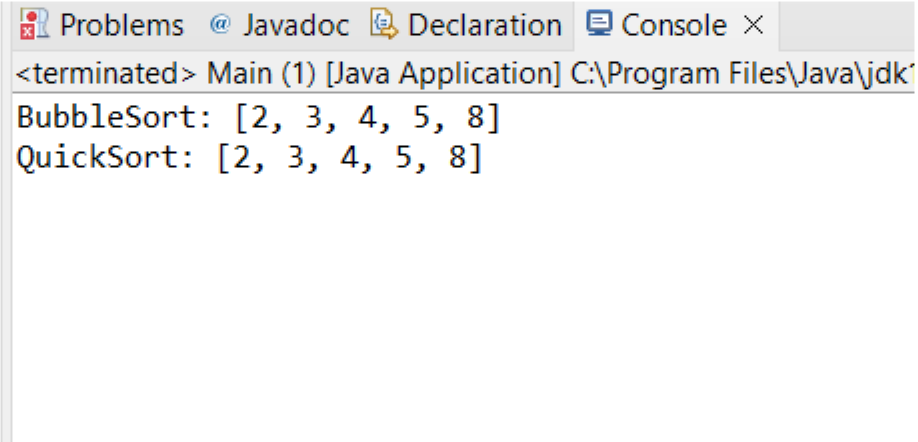
Develop a Context class that can use different SortingStrategy algorithms interchangeably to sort a collection of numbers.

Solution:

```
package Task;

public interface SortingStrategy {
    void sort(int[] numbers);
}

package Task;

public class BubbleSort 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 numbers[j] and numbers[j + 1]
                    int temp = numbers[j];
                    numbers[j] = numbers[j + 1];
                    numbers[j + 1] = temp;
                }
            }
        }
    }
}


package Task;

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

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

    private int partition(int[] array, int low, int high) {
        int pivot = array[high];
        int i = (low - 1);
        for (int j = low; j < high; j++) {
            if (array[j] < pivot) {
```

```java
                i++;
                int temp = array[i];
                array[i] = array[j];
                array[j] = temp;
            }
        }
        int temp = array[i + 1];
        array[i + 1] = array[high];
        array[high] = temp;
        return i + 1;
    }
}

package Task;

public class Context {
    private SortingStrategy strategy;

    public void setStrategy(SortingStrategy strategy) {
        this.strategy = strategy;
    }

    public void sortArray(int[] numbers) {
        if (strategy != null) {
            strategy.sort(numbers);
        } else {
            throw new IllegalStateException("Sorting strategy not set.");
        }
    }
}


package Task;
public class Main {
    public static void main(String[] args) {
        int[] numbers = {5, 3, 8, 4, 2};

        Context context = new Context();

        // Use BubbleSort strategy
        context.setStrategy(new BubbleSort());
        context.sortArray(numbers);
        System.out.println("BubbleSort: " + java.util.Arrays.toString(numbers));

        // Reset the array
        numbers = new int[]{5, 3, 8, 4, 2};

        // Use QuickSort strategy
        context.setStrategy(new QuickSort());
        context.sortArray(numbers);
        System.out.println("QuickSort: " + java.util.Arrays.toString(numbers));
    }
}
```

Output:



```
<terminated> Main (1) [Java Application] C:\Program Files\Java\jdk1
BubbleSort: [2, 3, 4, 5, 8]
QuickSort: [2, 3, 4, 5, 8]
```