In Java, data abstraction and encapsulation are two core concepts in object-oriented programming, each serving a different purpose for managing complexity and protecting data.

1. Data Abstraction

Abstraction is the process of hiding the complex implementation details and showing only the essential features of an object. It allows users to work at a high level, focusing on what an object does rather than how it does it. This is typically achieved through abstract classes or interfaces in Java.

For example, consider a `Car` class. When we drive a car, we only need to know how to start, stop, accelerate, and brake. We don't need to know how the engine works internally or how the braking system is designed. Abstraction helps in achieving this by exposing only the necessary operations to the user.

Example of Abstraction:

```java
abstract class Car {
    abstract void start();
    abstract void stop();
}

class Tesla extends Car {
    @Override
    void start() {
        System.out.println("Tesla started");
    }

    @Override
    void stop() {
        System.out.println("Tesla stopped");
    }
}

public class Main {
    public static void main(String[] args) {
        Car myCar = new Tesla();
        myCar.start();
        myCar.stop();
    }
}
```

Here, the `Car` class is abstract and hides the implementation details of `start()` and `stop()` methods. We know we can start and stop a car, but we don't know how it happens internally.

2. Encapsulation

Encapsulation is the technique of bundling data (attributes) and methods (functions) that operate on the data within a single unit, typically a class. Encapsulation also involves restricting access to certain details of an object, protecting the data from unauthorized access or modification. This is done by declaring the class fields as `private` and providing public `getter` and `setter` methods to access and modify them if needed.

Example of Encapsulation:

```java
class BankAccount {
    private double balance; // private variable to protect direct access

    public BankAccount(double initialBalance) {
        if (initialBalance > 0) {
            balance = initialBalance;
        }
    }

    public double getBalance() { // Getter to access balance
        return balance;
    }

    public void deposit(double amount) { // Method to modify balance
        if (amount > 0) {
            balance += amount;
        }
    }

    public void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
        }
    }
}

public class Main {
    public static void main(String[] args) {
        BankAccount account = new BankAccount(1000);
        account.deposit(500);
        account.withdraw(300);
        System.out.println("Current balance: " + account.getBalance());
    }
}
```

Here, the `BankAccount` class encapsulates the `balance` field, making it accessible only through `deposit`, `withdraw`, and `getBalance` methods. The `balance` field cannot be accessed or modified directly from outside the class, providing a controlled way to manage data.

Summary of Differences

Aspect	Data Abstraction	Encapsulation
Purpose	To show only the essential features, hiding unnecessary details.	To bind data and methods together and restrict direct access.
Focus	What an object does	How data is protected and accessed
Implementation	Achieved with abstract classes and interfaces	Achieved with `private` fields and `public` getter/setter methods
Example Usage	Defining a `Car` interface or abstract class with essential methods	Making `balance` in `BankAccount` private and providing methods to modify it

Abstraction helps in focusing on high-level operations, while encapsulation is about controlling access to data for security and protection.