Problem 1: Simple Encapsulation Example

Problem:
Create a class `Person` with private fields for `name` and `age`. Provide getter and setter methods for these fields. Ensure that the `age` cannot be negative.

Solution:

```
java
Copy code
class Person {
    private String name;
    private int age;

    // Getter for name
    public String getName() {
        return name;
    }

    // Setter for name
    public void setName(String name) {
        this.name = name;
    }

    // Getter for age
    public int getAge() {
        return age;
    }

    // Setter for age with validation
    public void setAge(int age) {
        if (age >= 0) {
            this.age = age;
        } else {
            System.out.println("Age cannot be negative");
        }
    }
}

public class Main {
    public static void main(String[] args) {
        Person person = new Person();
        person.setName("John");
        person.setAge(25);  // valid age
        System.out.println(person.getName() + " is " + person.getAge() + "
years old.");

        person.setAge(-5);  // invalid age
    }
}
```

Output:

```
csharp
```

```
Copy code
John is 25 years old.
Age cannot be negative
```

Problem 2: Encapsulation with Constructor

Problem:
Create a class `BankAccount` that holds a private balance field. Create a constructor to set an initial balance and provide methods to deposit and withdraw money, ensuring the balance cannot go below zero.

Solution:

```
java
Copy code
class BankAccount {
    private double balance;

    // Constructor to set the initial balance
    public BankAccount(double initialBalance) {
        if (initialBalance >= 0) {
            this.balance = initialBalance;
        } else {
            System.out.println("Initial balance cannot be negative");
        }
    }

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

    // Deposit method
    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
        } else {
            System.out.println("Deposit amount must be positive");
        }
    }

    // Withdraw method with validation
    public void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
        } else {
            System.out.println("Insufficient balance or invalid amount");
        }
    }
}

public class Main {
    public static void main(String[] args) {
        BankAccount account = new BankAccount(1000);
        System.out.println("Initial Balance: $" + account.getBalance());

        account.deposit(500);
        System.out.println("After deposit: $" + account.getBalance());

        account.withdraw(200);
        System.out.println("After withdrawal: $" + account.getBalance());

        account.withdraw(1500);  // invalid withdrawal
    }
}
```

Output:

```
bash
Copy code
Initial Balance: $1000.0
After deposit: $1500.0
After withdrawal: $1300.0
Insufficient balance or invalid amount
```

Problem 3: Encapsulation with Multiple Classes

Problem:
Create two classes, `Student` and `Course`. The `Student` class will have private fields for name and ID, and a method to enroll in a `Course`. The `Course` class will have private fields for the course name and instructor, with appropriate getters and setters.

Solution:

```
java
Copy code
class Course {
    private String courseName;
    private String instructor;

    public Course(String courseName, String instructor) {
        this.courseName = courseName;
        this.instructor = instructor;
    }

    // Getters
    public String getCourseName() {
        return courseName;
    }

    public String getInstructor() {
        return instructor;
    }
}

class Student {
    private String name;
    private int id;
    public Student(String name, int id) {
        this.name = name;
        this.id = id;
    }

    public void enrollInCourse(Course course) {
        System.out.println(name + " has enrolled in " +
course.getCourseName() + " taught by " + course.getInstructor());
    }
}

public class Main {
    public static void main(String[] args) {
        Course javaCourse = new Course("Java Programming", "Mr. Smith");
        Student student = new Student("Alice", 12345);

        student.enrollInCourse(javaCourse);
    }
}
```

Output:

```
csharp
Copy code
Alice has enrolled in Java Programming taught by Mr. Smith
```

Problem 4: Encapsulation in Real Life - Car Class

Problem:
Design a class `Car` that encapsulates the properties of a car, such as `make`, `model`, and `fuel level`. Add methods to `drive` the car (which reduces fuel) and `refuel` the car.

Solution:

```
java
Copy code
class Car {
    private String make;
    private String model;
    private double fuelLevel;

    // Constructor
    public Car(String make, String model, double fuelLevel) {
        this.make = make;
        this.model = model;
        if (fuelLevel >= 0) {
            this.fuelLevel = fuelLevel;
        } else {
            System.out.println("Fuel level cannot be negative");
        }
    }

    // Getters
    public String getMake() {
        return make;
    }

    public String getModel() {
        return model;
    }

    public double getFuelLevel() {
        return fuelLevel;
    }

    // Method to drive the car, which reduces fuel
    public void drive(double distance) {
        double fuelConsumed = distance * 0.1;  // assuming 10 km per liter
        if (fuelLevel >= fuelConsumed) {
            fuelLevel -= fuelConsumed;
            System.out.println("Drove " + distance + " km. Remaining fuel: "
+ fuelLevel + " liters.");
        } else {
            System.out.println("Not enough fuel to drive that distance.");
        }
    }

    // Method to refuel the car
    public void refuel(double liters) {
        if (liters > 0) {
            fuelLevel += liters;
            System.out.println("Refueled " + liters + " liters. Current fuel
level: " + fuelLevel + " liters.");
        } else {
            System.out.println("Invalid refuel amount.");
        }
    }
}

public class Main {
    public static void main(String[] args) {
        Car myCar = new Car("Toyota", "Corolla", 20);
        myCar.drive(50);  // valid drive
        myCar.refuel(10);  // refuel
        myCar.drive(200);  // not enough fuel
    }
}
```

Output:

```
yaml
Copy code
Drove 50.0 km. Remaining fuel: 15.0 liters.
Refueled 10.0 liters. Current fuel level: 25.0 liters.
Not enough fuel to drive that distance.
```