CHRIST (DEEMED TO BE UNIVERSITY)

Department of Computer Science

MSc – Artificial Intelligence and Machine Learning

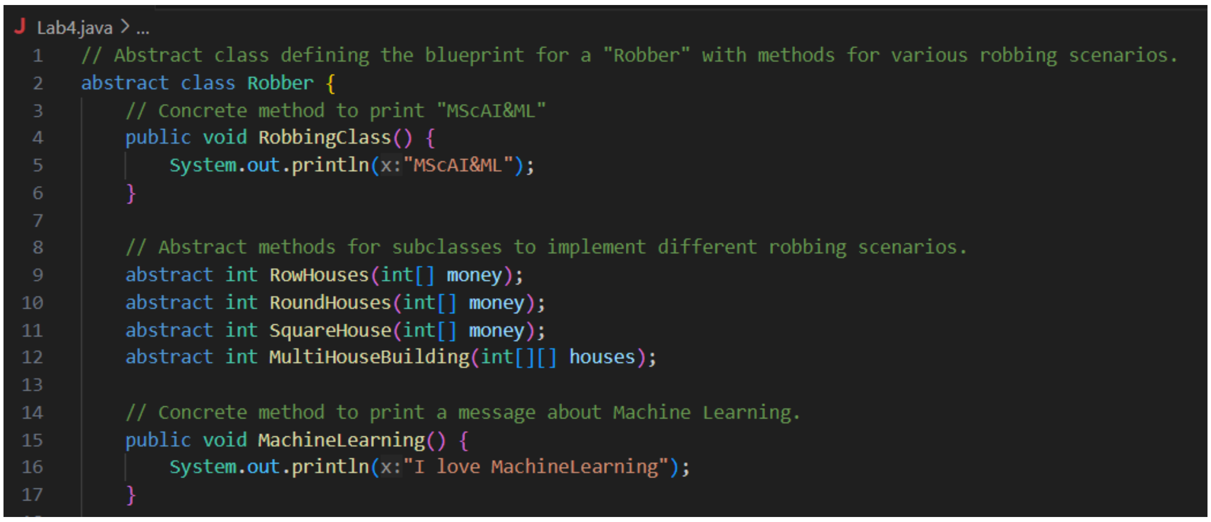
Name: Joel Joseph Motha **Reg. No:** 2448521

Course: Java Programming **Lab Program:** 4

Description

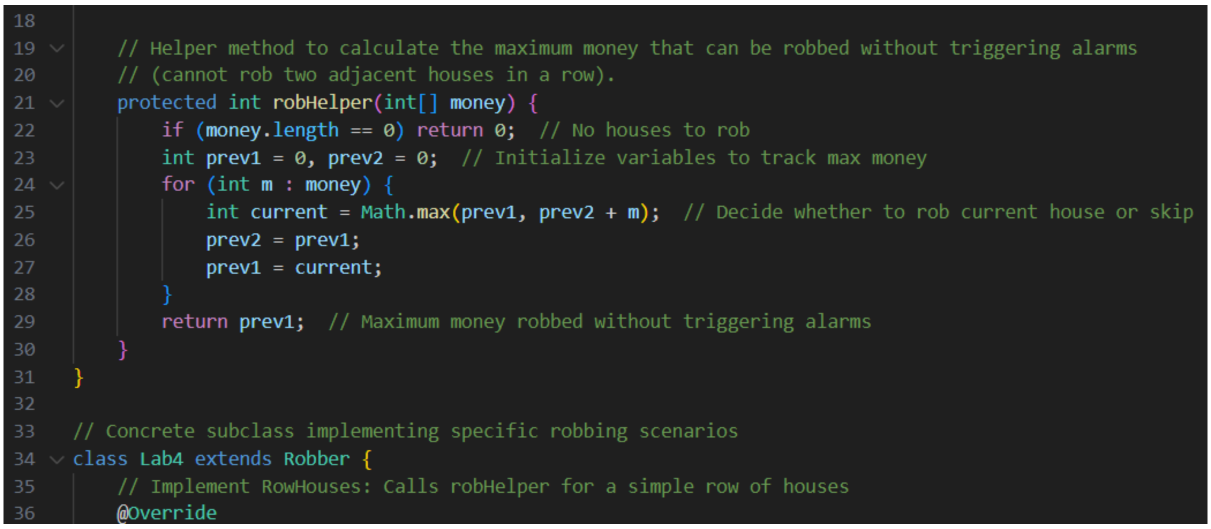
The code defines an abstract class Robber with methods for different robbing scenarios (RowHouses, RoundHouses, SquareHouse, MultiHouseBuilding). A concrete subclass JAVAProfessionalRobber implements these methods, using a helper function to calculate the maximum money that can be robbed without triggering alarms (by skipping adjacent houses). The program tests these methods with various inputs, calculating and printing the maximum money that can be robbed from a set of houses or buildings.

Code Screenshots

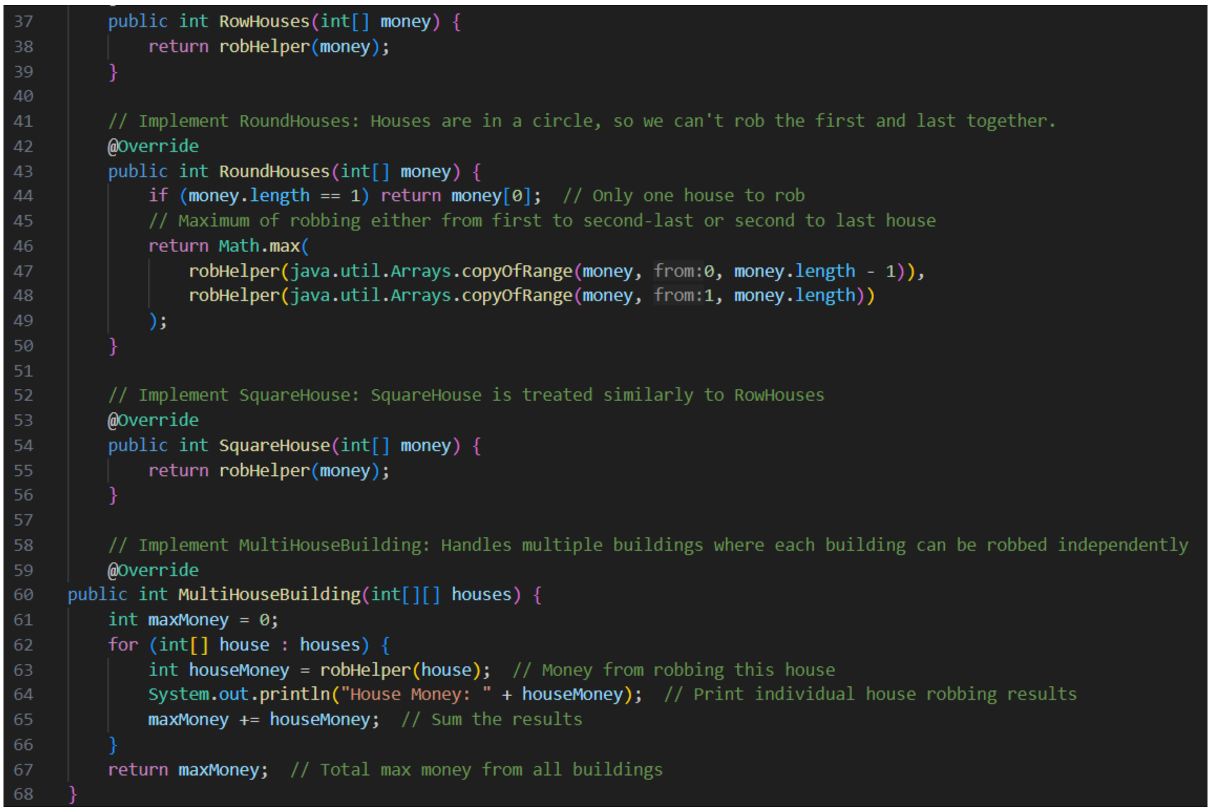


Lab4.java > ...

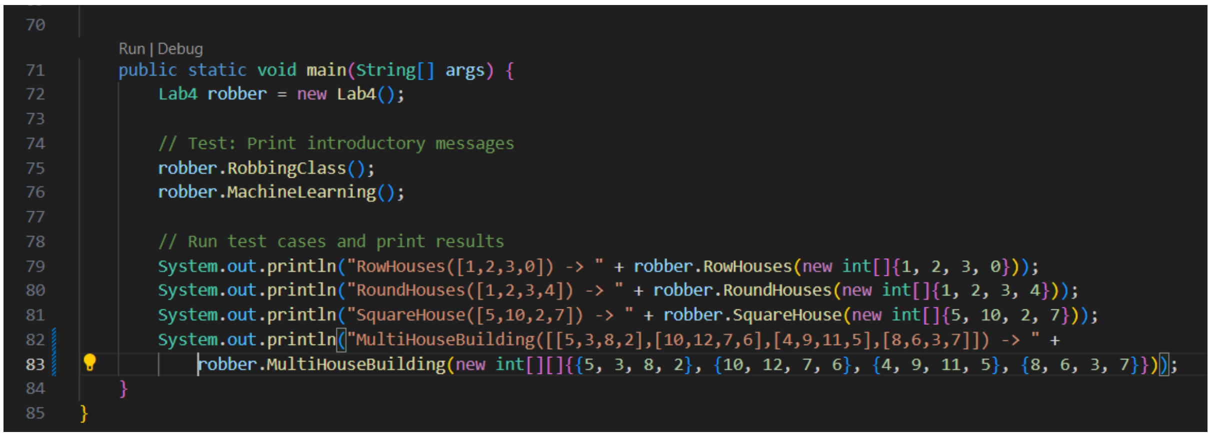
```java
// Abstract class defining the blueprint for a "Robber" with methods for various robbing scenarios.
abstract class Robber {
    // Concrete method to print "MScAI&ML"
    public void RobbingClass() {
        System.out.println(x:"MScAI&ML");
    }

    // Abstract methods for subclasses to implement different robbing scenarios.
    abstract int RowHouses(int[] money);
    abstract int RoundHouses(int[] money);
    abstract int SquareHouse(int[] money);
    abstract int MultiHouseBuilding(int[][] houses);

    // Concrete method to print a message about Machine Learning.
    public void MachineLearning() {
        System.out.println(x:"I love MachineLearning");
    }
```



```java

    // Helper method to calculate the maximum money that can be robbed without triggering alarms
    // (cannot rob two adjacent houses in a row).
    protected int robHelper(int[] money) {
        if (money.length == 0) return 0;  // No houses to rob
        int prev1 = 0, prev2 = 0;  // Initialize variables to track max money
        for (int m : money) {
            int current = Math.max(prev1, prev2 + m);  // Decide whether to rob current house or skip
            prev2 = prev1;
            prev1 = current;
        }
        return prev1;  // Maximum money robbed without triggering alarms
    }
}

// Concrete subclass implementing specific robbing scenarios
class Lab4 extends Robber {
    // Implement RowHouses: Calls robHelper for a simple row of houses
    @Override
```



```java
    public int RowHouses(int[] money) {
        return robHelper(money);
    }

    // Implement RoundHouses: Houses are in a circle, so we can't rob the first and last together.
    @Override
    public int RoundHouses(int[] money) {
        if (money.length == 1) return money[0];  // Only one house to rob
        // Maximum of robbing either from first to second-last or second to last house
        return Math.max(
            robHelper(java.util.Arrays.copyOfRange(money, from:0, money.length - 1)),
            robHelper(java.util.Arrays.copyOfRange(money, from:1, money.length))
        );
    }

    // Implement SquareHouse: SquareHouse is treated similarly to RowHouses
    @Override
    public int SquareHouse(int[] money) {
        return robHelper(money);
    }

    // Implement MultiHouseBuilding: Handles multiple buildings where each building can be robbed independently
    @Override
public int MultiHouseBuilding(int[][] houses) {
    int maxMoney = 0;
    for (int[] house : houses) {
        int houseMoney = robHelper(house);  // Money from robbing this house
        System.out.println("House Money: " + houseMoney);  // Print individual house robbing results
        maxMoney += houseMoney;  // Sum the results
    }
    return maxMoney;  // Total max money from all buildings
}
```



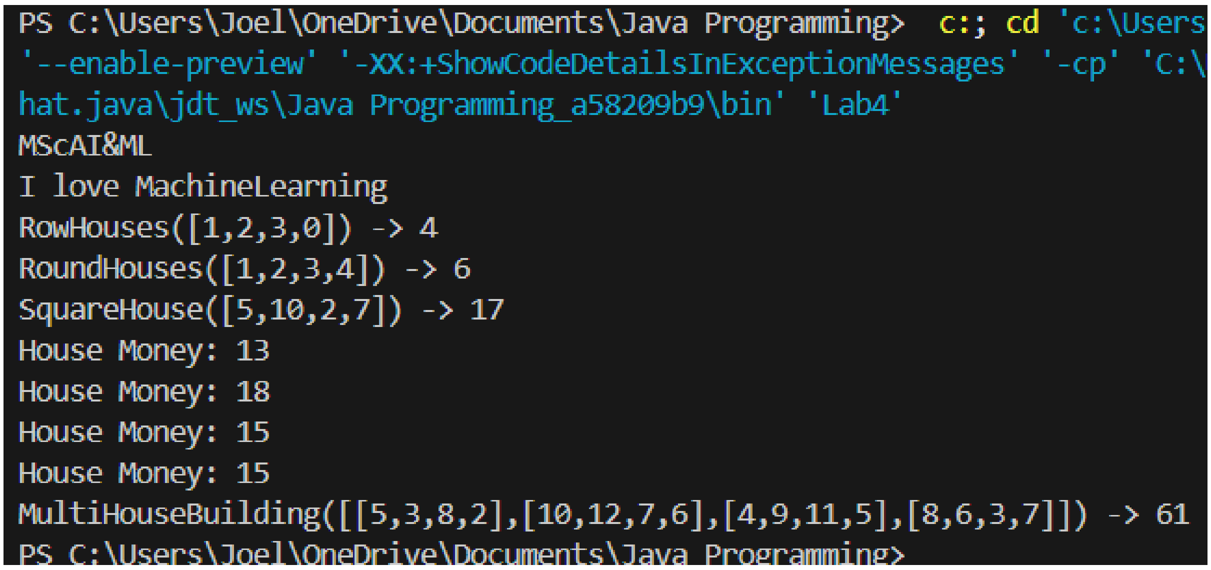
```java
    public static void main(String[] args) {
        Lab4 robber = new Lab4();

        // Test: Print introductory messages
        robber.RobbingClass();
        robber.MachineLearning();

        // Run test cases and print results
        System.out.println("RowHouses([1,2,3,0]) -> " + robber.RowHouses(new int[]{1, 2, 3, 0}));
        System.out.println("RoundHouses([1,2,3,4]) -> " + robber.RoundHouses(new int[]{1, 2, 3, 4}));
        System.out.println("SquareHouse([5,10,2,7]) -> " + robber.SquareHouse(new int[]{5, 10, 2, 7}));
        System.out.println("MultiHouseBuilding([[5,3,8,2],[10,12,7,6],[4,9,11,5],[8,6,3,7]]) -> " +
            robber.MultiHouseBuilding(new int[][]{{5, 3, 8, 2}, {10, 12, 7, 6}, {4, 9, 11, 5}, {8, 6, 3, 7}}));
    }
}
```

Output

```
MScAI&ML

I love MachineLearning

RowHouses([1,2,3,0]) -> 4

RoundHouses([1,2,3,4]) -> 6

SquareHouse([5,10,2,7]) -> 17

House Money: 13

House Money: 18

House Money: 15
```

House Money: 15

MultiHouseBuilding([[5,3,8,2],[10,12,7,6],[4,9,11,5],[8,6,3,7]]) -> 61



```
PS C:\Users\Joel\OneDrive\Documents\Java Programming>  c:; cd 'c:\Users
'--enable-preview' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\
hat.java\jdt_ws\Java Programming_a58209b9\bin' 'Lab4'
MScAI&ML
I love MachineLearning
RowHouses([1,2,3,0]) -> 4
RoundHouses([1,2,3,4]) -> 6
SquareHouse([5,10,2,7]) -> 17
House Money: 13
House Money: 18
House Money: 15
House Money: 15
MultiHouseBuilding([[5,3,8,2],[10,12,7,6],[4,9,11,5],[8,6,3,7]]) -> 61
PS C:\Users\Joel\OneDrive\Documents\Java Programming>
```