

## Problem 1:

### Problem Description:

In this problem, we are tasked with designing a custom Integer-like wrapper class named `MyInteger`. This class must contain an integer value and have many ways to operate on it, including checking for evenness, oddness, primality, and equality. It should also provide ways to convert the numeric characters in arrays and strings into integer values.

### Why does this problem matter?

This issue is important because it clarifies the foundations of Java class design and object-oriented programming. It gives a practical illustration of how a custom class might function inside, showing how to build one to encapsulate data and action. Additionally, it improves our comprehension of fundamental mathematical ideas and their real-world programming applications by implementing mathematical checks for evenness, oddness, and primality.

Understanding the concepts of encapsulation, developing custom data types, implementing techniques for data validation and manipulation, and applying these ideas in diverse programming jobs and projects are necessary to transfer the learning from this problem to other circumstances.

### Analysis:

### Design and Solution:

To address this problem, we design the `MyInteger` class with the following key components:

1. An `int` data field named `value` to store the integer value.
2. A constructor to create a `MyInteger` object for a specified `int` value.
3. Getter methods to retrieve the `int` value.
4. Methods for checking evenness, oddness, and primality.
5. Static methods for checking evenness, oddness, and primality for specified values.

6. Methods for equality checks.

7. Static methods for parsing numeric characters from arrays and strings into integer values.

Difficulties Encountered:

There could be a number of obstacles in the way of implementing this course. Several of these difficulties include:

- Ensuring the effectiveness and accuracy of the mathematical tests for evenness, oddness, and primality.
- Handling method overloads for various argument types, such as 'isEven(int)' and 'isEven(MyInteger)'.
- Correctly handling invalid inputs when parsing strings and characters.

Alternate Solutions:

There could be other approaches to solving this issue. The secret, though, is to create a class that complies with the guidelines. Different programmers may opt to implement specific methods in a different way or use various mathematical checking procedures, but the underlying structure of the "MyInteger" class should adhere to the specifications given.

We learn important lessons about object-oriented programming, problem-solving, and the significance of developing well-structured classes and methods by comprehending and overcoming these difficulties. The 'MyInteger' class acts as a base for implementing related concepts in different programming contexts.

Source Code:

```
package edu.northeastern.csye6200;

public class LAB5P1 {
    public static void main(String[] args) {
        MyInteger n1 = new MyInteger(7);
        MyInteger n2 = new MyInteger(24);

        System.out.println("n1 is even? " + n1.isEven());
        System.out.println("n1 is prime? " + n1.isPrime());
        System.out.println("15 is prime? " + MyInteger.isPrime(15));
    }
}
```

```

        char[] charArray = {'4', '3', '7', '8'};
        System.out.println("parseInt(char[]) for { '4', '3', '7', '8' } = " +
MyInteger.parseInt(charArray));

        String str = "4378";
        System.out.println("parseInt(String) for \"" + str + "\" = " +
MyInteger.parseInt(str));

        System.out.println("n2 is odd? " + n2.isOdd());
        System.out.println("45 is odd? " + MyInteger.isOdd(45));

        System.out.println("n1 is equal to n2? " + n1.equals(n2));
        System.out.println("n1 is equal to 5? " + n1.equals(5));
    }
}

```

```

class MyInteger {
    private int value;

    public MyInteger(int value) {
        this.value = value;
    }

    public int getValue() {
        return value;
    }

    public boolean isEven() {
        return value % 2 == 0;
    }

    public static boolean isEven(int n) {
        return n % 2 == 0;
    }

    public static boolean isEven(MyInteger o) {
        return o.isEven();
    }

    public boolean isOdd() {
        return value % 2 != 0;
    }

    public static boolean isOdd(int n) {
        return n % 2 != 0;
    }

    public static boolean isOdd(MyInteger o) {
        return o.isOdd();
    }

    public boolean isPrime() {
        if (value <= 1) {
            return false;
        }
    }
}

```

```

    }
    if (value <= 3) {
        return true;
    }
    if (value % 2 == 0 || value % 3 == 0) {
        return false;
    }
    for (int i = 5; i * i <= value; i += 6) {
        if (value % i == 0 || value % (i + 2) == 0) {
            return false;
        }
    }
    return true;
}

public static boolean isPrime(int num) {
    return new MyInteger(num).isPrime();
}

public static boolean isPrime(MyInteger o) {
    return o.isPrime();
}

public boolean equals(int anotherNum) {
    return value == anotherNum;
}

public boolean equals(MyInteger o) {
    return value == o.getValue();
}

public static int parseInt(char[] numbers) {
    int result = 0;
    for (char c : numbers) {
        if (Character.isDigit(c)) {
            result = result * 10 + Character.getNumericValue(c);
        }
    }
    return result;
}

public static int parseInt(String s) {
    int result = 0;
    for (int i = 0; i < s.length(); i++) {
        char c = s.charAt(i);
        if (Character.isDigit(c)) {
            result = result * 10 + Character.getNumericValue(c);
        }
    }
    return result;
}
}

```

## Screenshots of Sample Runs:



```
<terminated> LAB5P1 [Java Application] C:\Users\ADMIN\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_20.0.2.v202307180101\jre\bin
n1 is even? false
n1 is prime? true
15 is prime? false
parseInt(char[]) for { '4', '3', '7', '8' } = 4378
parseInt(String) for "4378" = 4378
n2 is odd? false
45 is odd? true
n1 is equal to n2? false
n1 is equal to 5? false
```

## Problem 2:

### Problem Description:

In this problem, we are tasked with creating a class called `RoomPeople` that can be used to record the number of people in different rooms of a building. The class has two attributes: `numberInRoom` to indicate the number of people in a specific room and `totalNumber` (a static variable) to indicate the total number of persons in all rooms. This class provides ways to enter and exit certain rooms, as well as ways to count individuals or groups of people in each area.

### Why does this problem matter?

This problem simulates circumstances where we need to handle data related to various things (in this case, rooms) and track changes in those entities (adding and removing people). It's crucial for software developers to understand how to package such data and provide solutions for handling it. This problem also illustrates the use of static variables to keep track of cumulative data shared by all instances of a class.

To adapt the knowledge gained from this problem to other contexts, object-oriented concepts such as encapsulation, data management, and method design must be applied to diverse areas where data needs to be structured and handled.

### Analysis:

### Design and Solution:

To solve this problem, we design the `RoomPeople` class with the following attributes and methods:

- `numberInRoom`: An attribute representing the number of people in a room.
- `totalNumber`: A static attribute representing the total number of people across all rooms.

### Methods:

- `addOneToRoom()`: Adds a person to the room and increases the value of `totalNumber`.
- `removeOneFromRoom()`: Removes a person from the room, ensuring that `numberInRoom` does not go below zero, and decreases the value of `totalNumber` as needed.
- `getNumber()`: Returns the number of people in the room.
- `getTotal()`: A static method that returns the total number of people across all rooms.

Difficulties Encountered:

While implementing this class, some potential challenges include:

- Ensuring that after a removal operation, the `numberInRoom` attribute is still non-negative.
- Maintaining proper updates to the static variable `totalNumber` as persons are added to or deleted from rooms.
- Correctly managing numerous instances of the `RoomPeople` class while changing the static `totalNumber` attribute.

Alternate Solutions:

Different data structures or data management techniques could be used as alternative solutions to this issue. However, the fundamental idea—encapsulating data and offering means of engaging with it—remains the same.

We learn more about data management and encapsulation in object-oriented programming by tackling these problems. This information can be used in a variety of situations where data tracking and manipulation are necessary, such as resource allocation or inventory management systems.

Source Code:

```
package edu.northeastern.csye6200;

public class LAB5P2 {
    public static void main(String[] args) {
        RoomPeople roomA = new RoomPeople();
    }
}
```

```

RoomPeople roomB = new RoomPeople();

System.out.println("Add two to room a and three to room b");
roomA.addOneToRoom();
roomA.addOneToRoom();
roomB.addOneToRoom();
roomB.addOneToRoom();
roomB.addOneToRoom();

System.out.println("Room a holds " + roomA.getNumber());
System.out.println("Room b holds " + roomB.getNumber());
System.out.println("Total in all rooms is " + RoomPeople.getTotal());

System.out.println("Remove two from both rooms");
roomA.removeOneFromRoom();
roomA.removeOneFromRoom();
roomB.removeOneFromRoom();
roomB.removeOneFromRoom();

System.out.println("Room a holds " + roomA.getNumber());
System.out.println("Room b holds " + roomB.getNumber());
System.out.println("Total in all rooms is " + RoomPeople.getTotal());

System.out.println("Remove two from room a (should not change the values)");
roomA.removeOneFromRoom();
roomA.removeOneFromRoom();

System.out.println("Room a holds " + roomA.getNumber());
System.out.println("Room b holds " + roomB.getNumber());
System.out.println("Total in all rooms is " + RoomPeople.getTotal());
}
}

class RoomPeople {
    private int numberInRoom;
    private static int totalNumber;

    public RoomPeople() {
        numberInRoom = 0;
    }

    public void addOneToRoom() {
        numberInRoom++;
        totalNumber++;
    }

    public void removeOneFromRoom() {
        if (numberInRoom > 0) {
            numberInRoom--;
            totalNumber--;
        }
    }

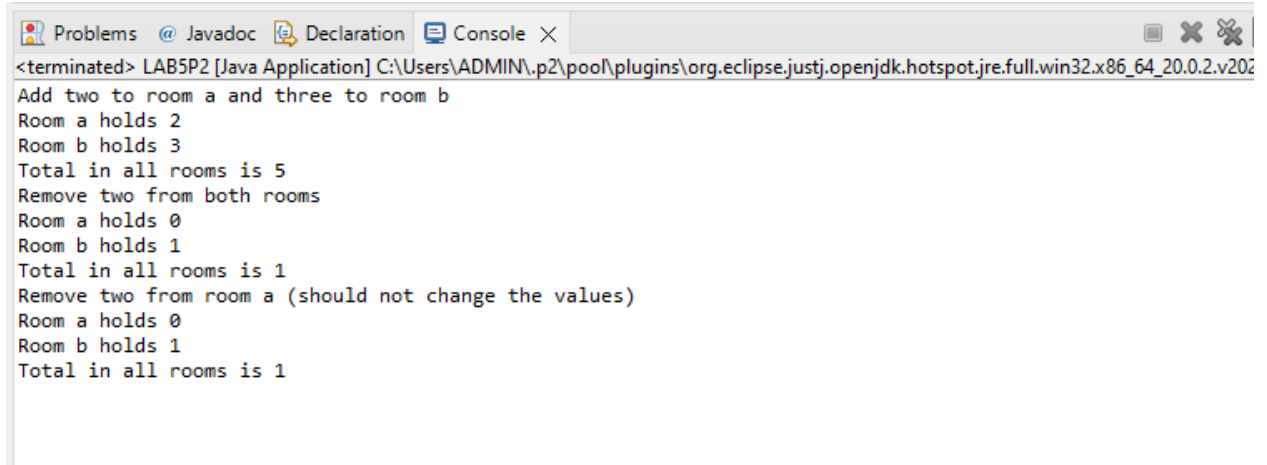
    public int getNumber() {
        return numberInRoom;
    }
}

```



```
    }  
  
    public static int getTotal() {  
        return totalNumber;  
    }  
}
```

Screenshots of Sample Runs:



```
<terminated> LAB5P2 [Java Application] C:\Users\ADMIN\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_20.0.2.v202  
Add two to room a and three to room b  
Room a holds 2  
Room b holds 3  
Total in all rooms is 5  
Remove two from both rooms  
Room a holds 0  
Room b holds 1  
Total in all rooms is 1  
Remove two from room a (should not change the values)  
Room a holds 0  
Room b holds 1  
Total in all rooms is 1
```

### Problem 3:

LAB5P3.java file:

```
package edu.northeastern.csye6200;

public class LAB5P3 {

    public static void main(String[] args) {
        Product milk = new Product("Milk", 3.7);
        Product bread = new Product("Bread", 2.25);
        Product eggs = new Product("Eggs", 4.3);

        Cart cart = new Cart();
        cart.addProduct(milk);
        cart.addProduct(eggs);

        System.out.println("Creating the below products");
        System.out.println(milk);
        System.out.println(bread);
        System.out.println(eggs);

        System.out.println("\nAdding Milk and Eggs to Cart");
        System.out.println(cart);
        System.out.println("Total Cart Value: $" + cart.getCartTotal());

        double payment = 10.0;
        double change = cart.calculateChange(payment);
        System.out.println("\nCustomer payment: $" + payment);
        System.out.println("Total Change: $" + change);
    }
}
```

Cart.java file:

```
package edu.northeastern.csye6200;

public class Cart {

    private StringBuilder products;
    private double cartTotal;

    public Cart() {
        products = new StringBuilder();
        cartTotal = 0.0;
    }

    public double getCartTotal() {
        return cartTotal;
    }
}
```

```

    }

    public void addProduct(Product product) {
        if (products.length() > 0) {
            products.append(", ");
        }
        products.append(product.getItemName());
        cartTotal += product.getPrice();
    }

    public double calculateChange(double payment) {
        return payment - cartTotal;
    }

    @Override
    public String toString() {
        return "Cart{ " + products + " }";
    }
}

```

Product.java file:

```

package edu.northeastern.csye6200;

public class Product {

    private String itemName;
    private double price;

    public Product(String itemName, double price) {
        this.itemName = itemName;
        this.price = price;
    }

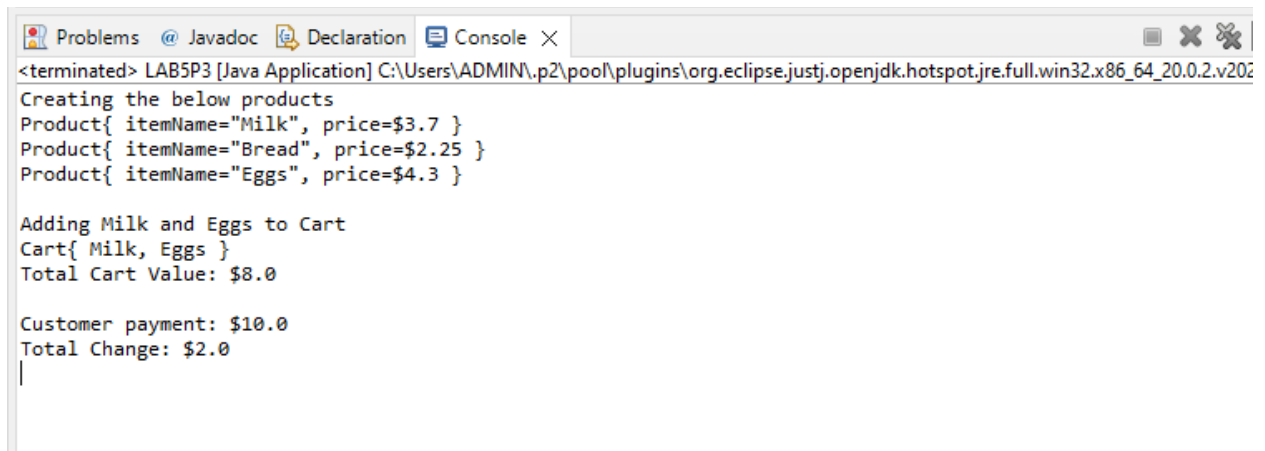
    public String getItemName() {
        return itemName;
    }

    public double getPrice() {
        return price;
    }

    @Override
    public String toString() {
        return "Product{ itemName=\"" + itemName + "\", price=$" + price + " }";
    }
}

```

## Screenshots of Sample Runs:



```
<terminated> LAB5P3 [Java Application] C:\Users\ADMIN\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_20.0.2.v202307180101\jre\bin\java.exe
Creating the below products
Product{ itemName="Milk", price=$3.7 }
Product{ itemName="Bread", price=$2.25 }
Product{ itemName="Eggs", price=$4.3 }

Adding Milk and Eggs to Cart
Cart{ Milk, Eggs }
Total Cart Value: $8.0

Customer payment: $10.0
Total Change: $2.0
|
```