

Lab 5 - Assignment 2 :- CSYE6200 - Harsh Sangani

Problem 1 :- Problem Description -

Designing a MyInteger Class for Number Analysis -

In this problem, we are tasked with creating a custom Java class called `MyInteger` that simulates the functionality of the built-in `Integer` class while also providing methods for analyzing numbers in various ways. The core functionality of this class involves storing an integer value, but it goes beyond that by allowing us to check if the number is even, odd, or prime, compare it with other integers, and perform conversions from character arrays or strings to integers.

Solving this problem extends to addressing a broader range of programming challenges:

- Custom Classes: The experience gained in designing a custom class can be applied to creating other custom data structures and utility classes.
- Algorithmic Thinking: The problem involves designing algorithms to determine whether a number is prime. This thought process can be transferred to other algorithmic problems.
- Method Design: Learning how to create and organize methods within a class is a transferrable skill, applicable in designing various software components.

Analysis -

Design/Algorithm Used:

To solve the problem of designing the `MyInteger` class, we need to consider various aspects of class design, method implementation, and algorithmic approaches for number analysis. Here's how we approach the problem:

1. Data Field: We define an integer data field `value` to store the number.
2. Constructor: We create a constructor to initialize the `value` field when a `MyInteger` object is created.
3. Getter Method: We provide a method to retrieve the value stored in `value`.

Conclusion:-

Solving this problem extends to addressing a broader range of programming challenges:

- Custom Classes: The experience gained in designing a custom class can be applied to creating other custom data structures and utility classes.
- Algorithmic Thinking: The problem involves designing algorithms to determine whether a number is prime. This thought process can be transferred to other algorithmic problems.

- Method Design: Learning how to create and organize methods within a class is a transferrable skill, applicable in designing various software components.

In summary, the initial solution fulfills the problem requirements but may benefit from enhancements in terms of algorithm efficiency, error handling, and additional functionality for more versatile number analysis. The choice of a better solution depends on the specific use cases and requirements.

Difficulties Encountered:-

1. Prime Number Checking: Implementing a robust primality testing algorithm can be challenging. We need to efficiently check whether a number is prime, which often involves iterating through possible divisors.
2. Character to Integer Conversion: Converting character arrays and strings to integers can be tricky, especially handling invalid inputs and edge cases.
3. Method Overloading: Understanding and correctly implementing method overloading, where multiple methods share the same name but differ in their parameter lists, can be complex.

4. Static vs. Instance Methods: Deciding when to use instance methods (methods that require an object) and static methods (methods that don't require an object) can be a design challenge.

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));
    }
}

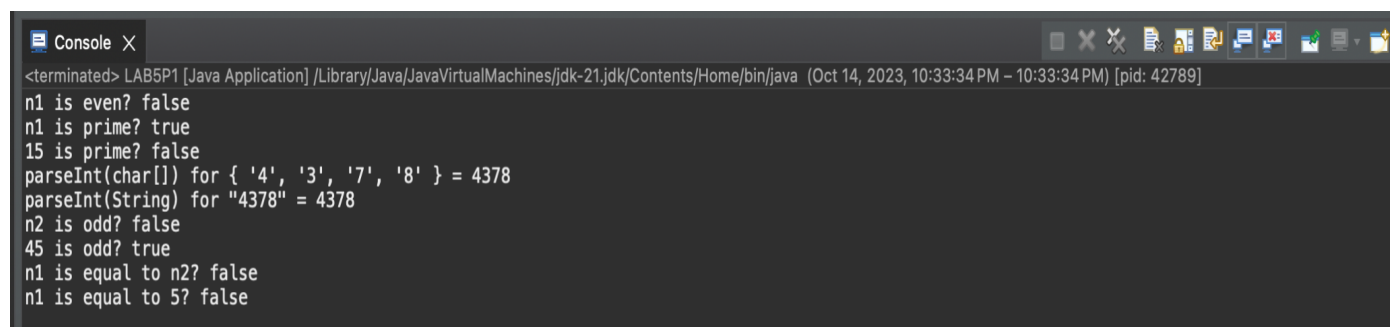
class MyInteger {
    private int value;
    public MyInteger(int value) {
        this.value = value;
    }
    public int getValue() {
        return value;
    }
    public boolean isEven() {
        return value % 2 == 0;
    }
    public static boolean isEven(int n) {
        return n % 2 == 0;
    }
    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] /Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java (Oct 14, 2023, 10:33:34 PM - 10:33:34 PM) [pid: 42789]
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 aim to create a Java class, RoomPeople, to accurately record and manage the number of people in various rooms within a building. The class has several attributes and methods, making it a practical exercise in object-oriented programming and encapsulation.

The Issue:

The issue we're addressing here is the need to keep track of the number of people in different rooms within a building, allowing us to add or remove individuals from each room, while also maintaining a record of the total number of people in the building. This problem is applicable in various real-world scenarios such as managing occupancy in hotels, conference centers, or any facility where the number of people in various rooms matters.

Significance and Learning:

- **Relevance:** Accurate monitoring of room occupancy is crucial for several practical purposes, including safety, resource allocation, and facility management. Learning how to create a class like `RoomPeople` is relevant and applicable in a wide range of situations.
- **Fundamental Object-Oriented Principles:** This problem provides an opportunity to understand fundamental object-oriented programming principles like encapsulation, abstraction, and static members. It's a foundation for more complex software design.
- **Transferable Knowledge:** The skills learned here can be easily applied to other software development scenarios where managing state and interactions between objects is important.
- **Safety and Efficiency:** Maintaining the number of people in each room and ensuring it doesn't go below zero is critical for safety, and efficient tracking of the total number of people is essential for efficient resource management.

Analysis :-

Design and Solution:

To solve the `RoomPeople` problem, we use a class-based object-oriented approach in Java. The `'RoomPeople'` class is designed with the following key attributes and methods:

Attributes:

- `'numberInRoom'`: Represents the number of people in a specific room.
- `'totalNumber'` (static): Represents the total number of people in all rooms, shared across all instances of the class.

Methods:

- `'addOneToRoom(int count)'`: Adds a specified number of people to the room, updating both the room count and the total count.
- `'removeOneFromRoom(int count)'`: Removes a specified number of people from the room, ensuring it does not go below zero, and also updates the total count.
- `'getNumber()'`: Retrieves the number of people in the room.

- `'getTotal()'` (static): Retrieves the total number of people across all rooms.

Conclusion :-

While this problem may seem straightforward, it touches upon the fundamentals of software development, making it a valuable exercise for those learning Java and object-oriented programming. The `RoomPeople` class and its methods can be extended and integrated into more complex systems, making it a stepping stone for solving real-world problems related to data management and tracking.

Difficulties Encountered:-

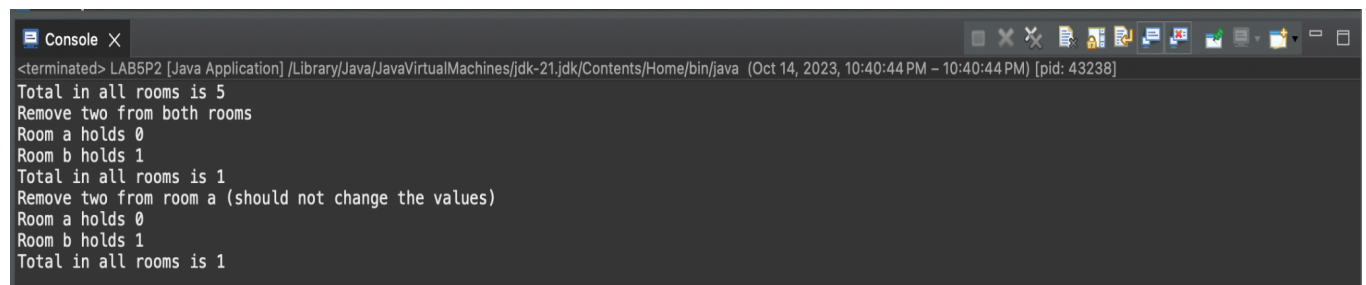
1. **Managing Total Count:** Ensuring that the `'totalNumber'` is correctly updated when adding or removing people from rooms can be tricky. This requires careful synchronization in multi-threaded scenarios to avoid race conditions.
2. **Input Validation:** Validating the input to `'addOneToRoom'` and `'removeOneFromRoom'` to prevent negative or invalid counts.
3. **Static vs. Instance Variables:** Deciding when and how to use static variables for shared state can be a bit challenging, as they are shared across all instances of the class.

Source Code :-

```
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] /Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java (Oct 14, 2023, 10:40:44 PM - 10:40:44 PM) [pid: 43238]
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
```

Problem3 :- Source Code:-

```
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;
    }

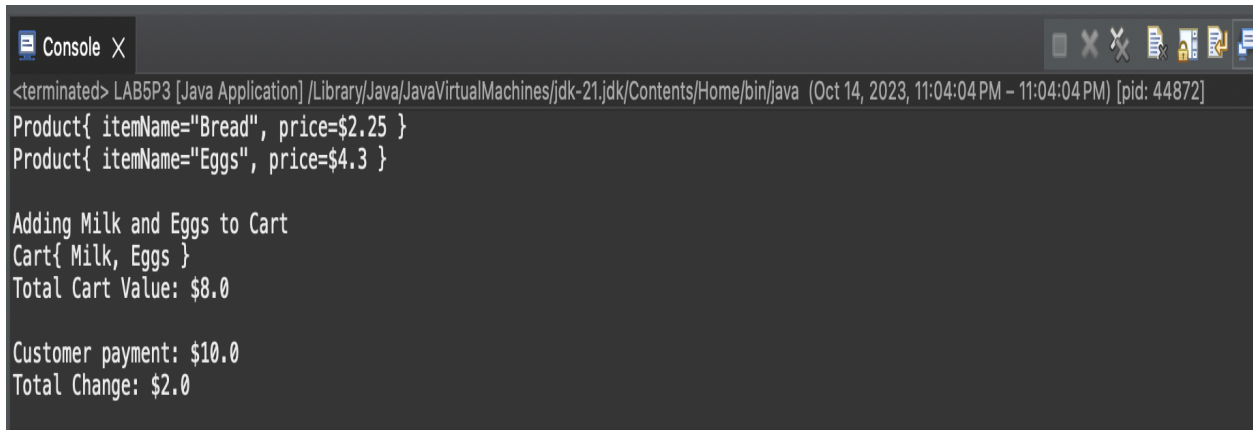
    public double getPrice() {
        return price;
    }

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

```

```
}  
}
```

Screenshots of sample runs :-



The screenshot shows a console window titled "Console X" with a dark background. The window contains the following text:

```
<terminated> LAB5P3 [Java Application] /Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java (Oct 14, 2023, 11:04:04 PM - 11:04:04 PM) [pid: 44872]  
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
```