**Vikas Meneni Lab 5 – Assignment 2 NUID:002309537**

**1.Problem 1**

**Problem Description:**

**The task is to design a custom wrapper class named MyInteger, which serves as an object-oriented representation of integer values, providing functionality to check if a value is even, odd, or prime, and includes static methods for validation. This exercise reinforces the principles of encapsulation and demonstrates the utility of managing primitive data types as objects. By implementing features like parseInt(char[]) and parseInt(String), we enhance our understanding of data conversion, which is essential in various programming contexts. Additionally, methods for comparing MyInteger objects highlight the importance of creating robust, reusable code. Overall, this class serves as an excellent opportunity to apply fundamental programming concepts and develop skills that are transferable to more complex challenges in software development.**

**Analysis:**

**The MyInteger class is designed to perform various operations on an integer value, serving as a simplified version of a wrapper class. Its structure is centered around providing functionalities to determine properties of the integer, such as whether it is even, odd, or prime.**

**The class includes a private integer field named value that holds the integer. A constructor initializes this field, and additional methods allow access to and manipulation of the stored value.**

**isEven(): This method checks if the integer is even by returning true when value % 2 == 0.**

**isOdd(): This method verifies if the integer is odd, returning true when value % 2 != 0.**

**isPrime(): This method evaluates the primality of the integer by checking for factors from 2 up to the square root of value, returning true if the number is prime.**

**The static methods isEven(int), isOdd(int), and isPrime(int) provide similar functionality for primitive integer values, enhancing flexibility.**

**The static methods isEven(MyInteger), isOdd(MyInteger), and isPrime(MyInteger) allow checks on instances of MyInteger.**

**The equals(int) and equals(MyInteger) methods assess whether the current object's value matches another integer or MyInteger object.**

**The methods parseInt(char[]) and parseInt(String) convert arrays of characters and strings into integer values without using Java’s built-in parsing capabilities**

**One challenge encountered during the implementation was ensuring accurate parsing of input values, particularly when converting character arrays and strings into integers. This process required careful handling to avoid errors, such as invalid characters or incorrect formats. Developing these functions without relying on Java’s Integer class necessitated a strong understanding of character manipulation and basic arithmetic.**

**A potential alternative approach could involve using Java’s Stream API to streamline the conversion process, improving both clarity and maintainability by adopting a more functional programming style. Additionally, incorporating error handling within the parsing methods would enhance robustness, enabling the class to handle invalid inputs, such as non-numeric characters, more effectively.**

**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[] chars = { '4', '3', '7', '8' };**

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

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

**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 boolean isOdd() {**

**return value % 2 != 0;**

**}**

**public boolean isPrime() {**

**if (value <= 1) return false;**

**for (int i = 2; i <= Math.*sqrt*(value); i++) {**

**if (value % i == 0) return false;**

**}**

**return true;**

**}**

**public static boolean isEven(int n) {**

**return n % 2 == 0;**

**}**

**public static boolean isOdd(int n) {**

**return n % 2 != 0;**

**}**

**public static boolean isPrime(int num) {**

**if (num <= 1) return false;**

**for (int i = 2; i <= Math.*sqrt*(num); i++) {**

**if (num % i == 0) return false;**

**}**

**return true;**

**}**

**public static boolean isEven(MyInteger o) {**

**return o.getValue() % 2 == 0;**

**}**

**public static boolean isOdd(MyInteger o) {**

**return o.getValue() % 2 != 0;**

**}**

**public static boolean isPrime(MyInteger o) {**

**return o.isPrime();**

**}**

**public boolean equals(int anotherNum) {**

**return this.value == anotherNum;**

**}**

**public boolean equals(MyInteger o) {**

**return this.value == o.getValue();**

**}**

**public static int parseInt(char[] numbers) {**

**int result = 0;**

**for (char c : numbers) {**

**result = result \* 10 + (c - '0');**

**}**

**return result;**

**}**

**public static int parseInt(String s) {**

**int result = 0;**

**for (int i = 0; i < s.length(); i++) {**

**result = result \* 10 + (s.charAt(i) - '0');**

**}**

**return result;**

**}**

**}**

**Screenshots of sample runs:**

**Screenshot :-**

**A screenshot of a computer

Description automatically generated**

**2.Problem 2**

**Problem Description:**

**The task is to create a Stock class that tracks details of stocks owned by an individual, including the ticker symbol, stock price (value), and quantity owned. The class uses a static attribute, totalValue, to maintain the total value of all stocks combined, calculated by multiplying the value and quantity of each stock. The class must include methods to set and retrieve the stock's quantity, value, and symbol, and update totalValue whenever stock details change. A parameterized constructor ensures that each stock has a valid, unchangeable symbol upon creation, while methods like getTotalValue() and updateTotalValue() help manage and track the overall stock portfolio value. This exercise emphasizes key object-oriented principles such as managing both individual object data and shared static data, highlighting the importance of consistent data updates across multiple stock objects.**

**Analysis:**

**The Stock class is crafted to effectively manage an individual's stock ownership, encapsulating essential details while simultaneously tracking the total value of all stocks owned. This class plays a vital role in investment management applications, where monitoring the performance and value of assets is crucial.**

**getTotalValue(): This method returns the cumulative value of all stocks, offering a convenient summary for users.**

**updateTotalValue(int oldValue, int oldQuantity, int newValue, int newQuantity): This method recalculates the static totalValue whenever there are changes in stock price or quantity, ensuring the overall total remains precise.**

**setQuantity(int quantity): This method updates the stock’s quantity and invokes updateTotalValue() to reflect this change in the overall value.**

**setValue(int value): Similar to setQuantity(), this method alters the stock's price and adjusts the total value accordingly.**

**getQuantity(): This method returns the current number of shares owned.**

**getValue(): This method retrieves the stock's current price.**

**getSymbol(): This method returns the symbol associated with the stock, providing an easy way to access its identifier.**

**toString(): This method offers a string representation of the Stock object, formatted to display the symbol, price, and quantity. For instance, it might return "Stock [AAPL, Value: $50, Quantity: 20]".**

**A notable challenge faced during the implementation of the Stock class was maintaining the accuracy of the totalValue in response to any updates to stock prices or quantities. This necessitated careful handling of the static methods to prevent any inconsistencies that could arise from simultaneous modifications.**

**Another alternative could be to implement a log system that records all changes made to stock values and quantities. This would allow users to track historical changes and revert to previous states if necessary. By maintaining a record of transactions, users can gain insights into their investment patterns**

**Top of Form**

**Bottom of Form**

**Source Code:**

**package edu.northeastern.csye6200;**

**public class LAB5P2 {**

**public static void main(String[] args) {**

**Stock appleStock = new Stock("AAPL");**

**appleStock.setValue(50);**

**appleStock.setQuantity(20);**

**Stock googleStock = new Stock("GOOGL");**

**googleStock.setValue(20);**

**googleStock.setQuantity(20);**

**System.*out*.println("Created the following stocks:");**

**System.*out*.println(appleStock);**

**System.*out*.println(googleStock);**

**System.*out*.println("Total Stock Value: " + Stock.*getTotalValue*());**

**System.*out*.println("\nUpdating the quantity of Apple stock");**

**appleStock.setQuantity(10);**

**System.*out*.println(appleStock);**

**System.*out*.println("Total Stock Value: " + Stock.*getTotalValue*());**

**System.*out*.println("\nUpdating the value of Google stock");**

**googleStock.setValue(100);**

**System.*out*.println(googleStock);**

**System.*out*.println("Total Stock Value: " + Stock.*getTotalValue*());**

**}**

**}**

**class Stock {**

**private final String symbol;**

**private int value;**

**private int quantity;**

**private static int *totalValue* = 0;**

**public Stock(String symbol) {**

**if (symbol == null || symbol.isEmpty()) {**

**throw new IllegalArgumentException("Stock symbol cannot be null or empty.");**

**}**

**this.symbol = symbol;**

**}**

**public static int getTotalValue() {**

**return *totalValue*;**

**}**

**public static void updateTotalValue(int oldValue, int oldQuantity, int newValue, int newQuantity) {**

***totalValue* -= (oldValue \* oldQuantity);**

***totalValue* += (newValue \* newQuantity);**

**}**

**public void setQuantity(int quantity) {**

**Stock.*updateTotalValue*(this.value, this.quantity, this.value, quantity);**

**this.quantity = quantity;**

**}**

**public void setValue(int value) {**

**Stock.*updateTotalValue*(this.value, this.quantity, value, this.quantity);**

**this.value = value;**

**}**

**public int getQuantity() {**

**return quantity;**

**}**

**public int getValue() {**

**return value;**

**}**

**public String getSymbol() {**

**return symbol;**

**}**

**public String toString() {**

**return "Stock [" + symbol + ", Value: $" + value + ", Quantity: " + quantity + "]";**

**}**

**}**

**Screenshots of sample runs:**

**A screenshot of a computer

Description automatically generated**

**3.Problem 3:**

**Source Code:**

**package edu.northeastern.csye6200;**

**public class LAB5P3 {**

**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.*totalNumber*);**

**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.*totalNumber*);**

**System.*out*.println("Remove two from room a");**

**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.*totalNumber*);**

**}**

**}**

**class RoomPeople {**

**int numberInRoom;**

**static int *totalNumber*;**

**public static int getTotal(){**

**return *totalNumber*;**

**}**

**public RoomPeople() {**

**numberInRoom = 0;**

**}**

**public void addOneToRoom(){**

**numberInRoom++;**

***totalNumber*++;**

**}**

**public void removeOneFromRoom(){**

**if(numberInRoom > 0) {**

**numberInRoom--;**

***totalNumber*--;**

**}**

**}**

**public int getNumber(){**

**return numberInRoom;**

**}**

**}**

**Screen shots of Sample Runs:-**

**A screenshot of a computer

Description automatically generated**