****

STIW5044 OBJECT ORIENTED PROGRAMMING DEVELOPMENT

ASSIGNMENT 2

GROUP MEMBERS

GOBEMATHAN SUBRAMANIAN (822710)

AHMAD HANNAN BIN ISA (826109)

1. Specify and implement the following methods with appropriate specifications (that contain REQUIRES (if applicable), MODIFIES (if applicable) and EFFECTS clauses):
   1. public static int smallValue(int [] arrayNum) that returns the smallest integer for a non-empty array of arrayNum.

**SOLUTIONS:**

.

REQUIRES: True

EFFECTS: If arrayNum has at least one element, returns the smallest value of arrayNum.

* 1. public static boolean isPrime (int num) that determines whether an integer num is prime.

**SOLUTIONS:**

EFFECTS: Returns true if the specifications is a prime number.

1. Modify the implementation of both methods in Question 1 as separate complete programs that each checks for an appropriate type of exceptions that might be thrown during program execution. Identify either the specified type of exception is checked or unchecked exception. You are required to prompt a user to enter the value(s) of its argument(s).

**SOLUTIONS (**Q1a**):**

|  |
| --- |
| package assignment2;  import java.util.NoSuchElementException;  import java.util.Scanner;  /\*\*  \*  \* @author gobe  \*/  public class Assignment2 {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) throws Exception {  //try catch block to handle exceptions  try{  Scanner input = new Scanner(System.in);  int [] numbers = new int [5]; // Create an array of five values  // Prompt user to enter five numbers  System.out.print("Enter five numbers: ");  for (int i = 0; i < numbers.length; i++)  numbers[i] = input.nextInt();  // To show the smallest value from input 5 numbers  System.out.println("Smallest value: "+ smallValue(numbers)) ;  }  catch (NoSuchElementException e){    System.out.println("Exception thrown " + e);  }  }  /\*\* Method smallValue returns the smallest values\*/  public static int smallValue(int[] arrayNum) {    if (arrayNum.length <= 1)  return 0;  int min = arrayNum[0];  for (int i = 1; i < arrayNum.length; i++) {  if (arrayNum[i] < min) {  min = arrayNum[i];  }  }  return min;  }  } |
| run:  Enter five numbers: 5  66  x  Exception thrown java.util.InputMismatchException  BUILD SUCCESSFUL (total time: 5 seconds) |
| run:  Enter five numbers: 56  6  8  1  22  Smallest value: 1  BUILD SUCCESSFUL (total time: 13 seconds) |

**SOLUTIONS (**Q1b**):**

|  |
| --- |
| package assignment2;  import java.util.Scanner;  /\*\*  \*  \* @author gobe  \*/  public class Assignment2b {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  Scanner input= new Scanner(System.in);  // Prompt user to enter any numbers  System.out.println("Enter any number:");  //try catch block to handle exceptions  try{  int num=input.nextInt();  if (isPrime(num)){  System.out.println(num + " is a Prime Number");  }  else{  System.out.println(num + " is not a Prime Number");  }  }    catch (Exception e){  System.out.println("Exception thrown "+ e);  }  }  /\*\* Method isPrime returns whether the value is a prime or not\*/  public static boolean isPrime (int num){  if (num<=1){  return false;  }  for (int i=2;i<num; i++){  if (num%i ==0){  return false;  }  }  return true;  }  } |
| run:  Enter any number:  test  Exception thrown java.util.InputMismatchException  BUILD SUCCESSFUL (total time: 3 seconds) |
| run:  Enter any number:  2.34  Exception thrown java.util.InputMismatchException  BUILD SUCCESSFUL (total time: 3 seconds) |
| run:  Enter any number:  3  3 is a Prime Number  BUILD SUCCESSFUL (total time: 2 seconds) |
| run:  Enter any number:  24  24 is not a Prime Number  BUILD SUCCESSFUL (total time: 3 seconds) |

1. A specification for a procedure that computes the sum of the elements in an array of integers might require a non-empty array, return 0 if the array is empty, or throw an exception if the array is empty. Discuss which alternative is best and provide the specification for the procedure.

**SOLUTIONS:**

The best alternative for the procedure that computes the sum of elements in an array of integers is by “throwing an exception” if the array is empty. This is because an empty list or return 0 are not meaningful and different error cases can’t be reflected with the stated specifications. By throwing exceptions, the procedure to compute will be more structured in dealing with errors that might occur during the run of the program. For this case, the Java method “REQUIRES” user to pass certain values as array list must not be empty, therefore, if the array list is empty then exception must be thrown. Some of exceptions that can be thrown for the stated procedure:

a) IllegalArgumentException: If the arrays are not of the same type and different length. (Checked)

b) ArithmeticException: If the results overflow an int type variable. (Unchecked)

c) IOException: Might occur if too many items. (Checked)

d) NoSuchElementException: Input errors. (Unchecked)

4. You need to propose a type Matrix with some of operations such as to add and multiply matrices and to invert a matrix. You are allowed to include additional operations so that the type provides adequate operations. These matrices are mutable. Specify and implement a class for Matrix with complete specifications (include with exceptions if applicable). Also, give the rep invariant and abstraction function for the type.

**SOLUTIONS:**

|  |
| --- |
| public class IntOperation extends Number {  //Overview: A util Class for data type Int for its operation  //mutable : value of Int change when any operation is called and default value x will be added  private Integer x;  public IntOperation() {  }  @Override  public int intValue() {  return x;  }  @Override  public long longValue() {  return Long.valueOf(this.x);  }  @Override  public float floatValue() {  //EFFECTS : will not mutate x value, only return the float value of x  return Float.valueOf(this.x);  }  @Override  public double doubleValue() {  //EFFECTS : will not mutate x value, only return the double value of x  return Double.valueOf(this.x);  }  public Integer getX() {  return this.x;  }  public void setX(int x) {  this.x = x;  }  public Integer add(int y) {  //MODIFIES: will add value y into current x value  //EFFECTS : will mutate the initial value of x  return this.x = this.x + y; }  public Integer substract(int y) {  //MODIFIES: will substract value y into current x value  //EFFECTS : will mutate the initial value of x  return this.x = this.x - y;  }  public Integer multiply(int y) {  //MODIFIES: will multiply value y into current x value  //EFFECTS : will mutate the initial value of x  return this.x = this.x \* y;  }  @Override  public String toString() {  return "IntOperation{" +  "x=" + x +  '}';  }  } |
| public class TestIntOperation {  public static void main(String[] args) {  IntOperation operation = new IntOperation();  int num = 12;  int number = 15;  operation.setX(num);  System.out.println("Adding Operation with "+number + " = "+operation.add(number));  System.out.println("Multiply Operation with "+number + " = "+operation.multiply(number));  System.out.println("Substract Operation with "+number + " = "+operation.substract(number));  System.out.println("Float value " + " = "+operation.floatValue());  System.out.println("Double value " + " = "+operation.doubleValue());  System.out.println(operation.toString());  }  } |
| run:  Adding Operation with 15 = 27  Multiply Operation with 15 = 405  Substract Operation with 15 = 390  Float value = 390.0  Double value = 390.0  IntOperation{x=390} |

5. Modify the implementation of the Matrix class that has been created in Question 4, so that the concept of iteration abstraction can be applied. To do so, provide the class with appropriate iterator method (s). Then, test the class to ensure each method has properly functioned.

**SOLUTIONS:**

|  |
| --- |
| import java.util.ArrayList;  import java.util.List;  /\*\*  \* @author Hannan  \*/  public class IntSetOperation extends ArrayList<Integer> {  // overview : IntSetOperation is a Utils to do operation on unbounded set of Integers  //mutable : state of Object can be change  private List<Integer> intSet;  public IntSetOperation() {  intSet = new ArrayList<Integer>();  }  public IntSetOperation(List<Integer> intSet) {  this.intSet = intSet;  }  public List<Integer> getIntSet() {  return intSet;  }  public void setIntSet(List<Integer> intSet) {  this.intSet = intSet;  }  @Override  public int size() {  return intSet.size();  }  @Override  public boolean isEmpty() {  return intSet.isEmpty();  }  @Override  public boolean contains(Object o) {  return intSet.contains(o);  }  @Override  public int indexOf(Object o) {  return intSet.indexOf(o);  }  @Override  public int lastIndexOf(Object o) {  return intSet.lastIndexOf(o);  }  @Override  public Object clone() {  return super.clone();  }  @Override  public Integer get(int index) {  return super.get(index);  }  @Override  public Integer set(int index, Integer element) {  //EFFECT: replace an Integer element on intset on specific index  return super.set(index, element);  }  @Override  public boolean add(Integer integer) {  //EFFECT: add new element into intSet  return intSet.add(integer);  }  @Override  public Integer remove(int index) {  //EFFECT : remove integer on specific index  return intSet.remove(index);  }  @Override  public void clear() {  //EFFECT : clear all Integer value  intSet.clear();  }  public Integer sums() throws NullPointerException{  //EFFECTS: if intSet is not null will iterate all the element and sum it one by one  return intSet.stream().mapToInt(Integer::intValue).sum();  }  public Integer multiply() throws NullPointerException{  return intSet.stream().mapToInt(Integer::intValue).reduce(1,Math::multiplyExact);  }  }  import java.util.ArrayList;  import java.util.List;  public class TestIntSetOperation {  public static void main(String[] args) {  List<Integer> integerList = new ArrayList<Integer>();  integerList.add(2);  integerList.add(5);  integerList.add(8);  integerList.add(10);  integerList.add(5);  integerList.add(6);  integerList.add(5);  integerList.add(8);  integerList.add(1);  IntSetOperation setOperation = new IntSetOperation(integerList);  System.out.println("Size of the integerList = "+setOperation.size());  System.out.println("Last Index of element valued 5 = "+setOperation.lastIndexOf(5));  System.out.println("Sum of IntegerList = "+ setOperation.sums());  System.out.println("Sum of multiplied value in IntegerList = "+ setOperation.multiply());  System.out.println("Value of integerList at index no 7 is " + integerList.get(7));  integerList.set(7,6);  System.out.println("New Value of integerList at index no 7 is "+ integerList.get(7));  integerList.clear();  System.out.println("New Size of integerList after clear : " +setOperation.size());  }  } |
|  |
| run:  Size of the integerList = 9  Last Index of element valued 5 = 6  Sum of IntegerList = 50  Sum of multiplied value in IntegerList = 960000  Value of integerList at index no 7 is 8  New Value of integerList at index no 7 is 6  New Size of integerList after clear : 0 |

6. Discuss about the following statement:

“A matrix abstraction ought not to be mutable”

Do you agree or disagree with the statement?

Abstraction in a general sense is a conceptual process which defines general rules and concepts derived on the usage and classification of specific example or behavior.

Conceptual abstractions is formed by observing a phenomenon and selecting any important aspect which are relevant on any purpose. In OOP concept, the main goal abstraction into handle complexity by hiding the implementation details from the user. This will allow user to enable more complex logic on the top of the provided abstraction without a need to understand the hidden complexity.

Object in OOP provide abstraction that hides the internal implementation details. User just need to know on which method to call and which parameter to pass to trigger a specific operation.

Thus, a matrix abstraction ought not to be mutable? To answer this, we need to understand the purpose of abstraction in OOP which to provide and standard concept on how the matrix should behave. Abstraction in higher level is to separate categorical concept related to computing problems. By doing this any program code does not have to depend on specific details of supporting applications, operating system nor hardware.

Matrix abstraction should not be mutable so others programmer or user can take advantage of another programmer’s work with only an abstract understanding of the implementation of other’s work.