Name : Abhimanyu Karia

Roll No : 202201435

Question 2

1) Armstrong Number:

Original Code:

```
// Armstrong Number
class Armstrong {
   public static void main(String args[]) {
      int num = Integer.parseInt(args[0]);
      int n = num; // use to check at last time
      int check = 0, remainder;
      while (num > 0) {
         remainder = num / 10;
         check = check + (int) Math.pow(remainder, 3);
         num = num % 10;
      }
      if (check == n)
         System.out.println(n + " is an Armstrong Number");
      else
         System.out.println(n + " is not a Armstrong Number");
   }
}
```

I. Errors in the code:

1. **Line 9: remainder = num / 10;**
    - **This line mistakenly divides the number by 10 to extract the last digit. It should use num % 10 to get the last digit.**
2. **Line 12: num = num % 10;**
    - **This line intends to remove the last digit but is using the modulus operator incorrectly. It should be num = num / 10;.**
3. **Missing closing bracket:**

- The code should include a closing bracket to properly finish the method.

II. Breakpoints:

- **Initial values: Check the variables num, check, and remainder at the start.**
- **Value updates: Observe how remainder changes after each division.**
- **Progression of num: Track how num is modified after each iteration.**

III. Steps to fix the errors:

1. **Correct operations:**
   - **Change remainder = num / 10 to remainder = num % 10 on line 9.**
   - **Update num = num % 10 to num = num / 10 on line 12.**

IV. FIXED CODE:

```
class Armstrong {
   public static void main(String args[]) {
      int num = Integer.parseInt(args[0]);
      int n = num; // use to check at the end
      int check = 0, remainder;

      while (num > 0) {
         remainder = num % 10; // Extract the last digit
         check = check + (int) Math.pow(remainder, 3); // Add the cube of the digit
         num = num / 10; // Remove the last digit
      }

      if (check == n)
         System.out.println(n + " is an Armstrong Number");
      else
         System.out.println(n + " is not an Armstrong Number");
   }
}
```

---

2) GCD and LCM:

Original Code:

```
public class GCD_LCM {
   static int gcd(int x, int y) {
      int r = 0, a, b;
      a = (x > y) ? y : x; // a is greater number
      b = (x < y) ? x : y; // b is smaller number

      r = b;
      while (a % b == 0) // Error replace it with while(a % b != 0)
      {
         r = a % b;
         a = b;
         b = r;
      }
      return r;
   }

   static int lcm(int x, int y) {
      int a;
      a = (x > y) ? x : y; // a is greater number
      while (true) {
         if (a % x != 0 && a % y != 0)
            return a;
         ++a;
      }
   }

   public static void main(String args[]) {
      Scanner input = new Scanner(System.in);
      System.out.println("Enter the two numbers: ");
      int x = input.nextInt();
      int y = input.nextInt();
```

```
        System.out.println("The GCD of two numbers is: " + gcd(x, y));
        System.out.println("The LCM of two numbers is: " + lcm(x, y));
        input.close();
    }
}
```

I. Errors in the code:

1. **GCD Calculation (Line 13):**
    - **The condition while(a % b == 0) is flawed. It should be while(a % b != 0) to ensure the loop continues until the remainder is zero.**
2. **LCM Calculation (Line 24):**
    - **The condition if(a % x != 0 && a % y != 0) is incorrect. It should be if(a % x == 0 && a % y == 0).**

II. Breakpoints:

- **Line 13: Check the logic within the GCD loop.**
- **Line 24: Verify the condition for LCM computation.**
- **Line 31: Confirm the correctness of the final GCD and LCM values.**

III. Steps to fix the errors:

1. **Fix the GCD calculation by updating the while loop condition.**
2. **Correct the LCM condition in the if statement.**

IV. FIXED CODE:

```
import java.util.Scanner;


public class GCD_LCM {
    static int gcd(int x, int y) {
        int r = 0, a, b;
        a = (x > y) ? x : y; // Assign the greater number
        b = (x < y) ? x : y; // Assign the smaller number


        while (a % b != 0) { // Continue until remainder is 0
            r = a % b;
            a = b;
```

```
            b = r;
        }
        return r; // The GCD is the last non-zero remainder
    }

    static int lcm(int x, int y) {
        int a = (x > y) ? x : y;
        while (true) {
            if (a % x == 0 && a % y == 0) // The least common multiple is divisible by both
                return a;
            ++a;
        }
    }

    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter the two numbers: ");
        int x = input.nextInt();
        int y = input.nextInt();

        System.out.println("The GCD of two numbers is: " + gcd(x, y));
        System.out.println("The LCM of two numbers is: " + lcm(x, y));
        input.close();
    }
}
```

---

3) Knapsack:

Original Code:

```
public class Knapsack {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);   // number of items
```

```
    int W = Integer.parseInt(args[1]);   // maximum weight of knapsack

    int[] profit = new int[N + 1];
    int[] weight = new int[N + 1];

    // generate random instance, items 1..N
    for (int n = 1; n <= N; n++) {
       profit[n] = (int) (Math.random() * 1000);
       weight[n] = (int) (Math.random() * W);
    }

    // opt[n][w] = max profit of packing items 1..n with weight limit w
    int[][] opt = new int[N + 1][W + 1];
    boolean[][] sol = new boolean[N + 1][W + 1];

    for (int n = 1; n <= N; n++) {
       for (int w = 1; w <= W; w++) {
          int option1 = opt[n++][w]; // This line contains an error

          int option2 = Integer.MIN_VALUE;
          if (weight[n] > w) option2 = profit[n - 2] + opt[n - 1][w - weight[n]]; // Incorrect profit
reference

          // select better of two options
          opt[n][w] = Math.max(option1, option2);
          sol[n][w] = (option2 > option1);
       }
    }

    // determine which items to take
    boolean[] take = new boolean[N + 1];
```

```
    for (int n = N, w = W; n > 0; n--) {
      if (sol[n][w]) {
        take[n] = true;
        w = w - weight[n]; // Logic may lead to issues
      } else {
        take[n] = false;
      }
    }

    // print results
    System.out.println("item" + "\t" + "profit" + "\t" + "weight" + "\t" + "take");
    for (int n = 1; n <= N; n++) {
      System.out.println(n + "\t" + profit[n] + "\t" + weight[n] + "\t" + take[n]);
    }
  }
}
```

I. Errors in the code:

1. **Line 20: int option1 = opt[n++][w];**
    - **The post-increment operator n++ causes n to increase prematurely, which may lead to an out-of-bounds error. This should be replaced with opt[n][w].**
2. **Line 24: option2 = profit[n - 2] + opt[n - 1][w - weight[n]];**
    - **The reference to profit[n - 2] is incorrect. It should simply use profit[n].**
3. **Line 32: Update logic for take[n].**
    - **The logic inside if (sol[n][w]) should prevent weight from becoming negative.**

II. Breakpoints:

- **Line 20: Check how option1 is being calculated.**
- **Line 24: Confirm the logic behind option2.**
- **Line 32: Ensure the items are correctly selected in the loop.**

III. Steps to fix the errors:

1. **Fix the increment error by removing n++ in line 20.**
2. **Correct the profit reference in line 24 by changing profit[n - 2] to profit[n].**

3. Adjust the weight logic in line 32 to prevent errors.

IV. FIXED CODE:

```
public class Knapsack {
   public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);
      int W = Integer.parseInt(args[1]);

      int[] profit = new int[N + 1];
      int[] weight = new int[N + 1];

      for (int n = 1; n <= N; n++) {
         profit[n] = (int) (Math.random() * 1000);
         weight[n] = (int) (Math.random() * W);
      }

      int[][] opt = new int[N + 1][W + 1];
      boolean[][] sol = new boolean[N + 1][W + 1];

      for (int n = 1; n <= N; n++) {
         for (int w = 1; w <= W; w++) {
            int option1 = opt[n][w]; // Fixed index access
            int option2 = Integer.MIN_VALUE;

            if (weight[n] <= w) // Ensure the item can be taken
               option2 = profit[n] + opt[n - 1][w - weight[n]]; // Correct profit index

            opt[n][w] = Math.max(option1, option2);
            sol[n][w] = (option2 > option1);
         }
      }
```

```
        boolean[] take = new boolean[N + 1];
        for (int n = N, w = W; n > 0; n--) {
            if (sol[n][w]) {
                take[n] = true;
                w -= weight[n]; // Correct weight update
            } else {
                take[n] = false;
            }
        }

        System.out.println("item" + "\t" + "profit" + "\t" + "weight" + "\t" + "take");
        for (int n = 1; n <= N; n++) {
            System.out.println(n + "\t" + profit[n] + "\t" + weight[n] + "\t" + take[n]);
        }
    }
}
```

4) Magic Number:

Original Code:

```
// Program to check if number is Magic number in JAVA
import java.util.*;
public class MagicNumberCheck {
    public static void main(String args[]) {
        Scanner ob = new Scanner(System.in);
        System.out.println("Enter the number to be checked.");
        int n = ob.nextInt();
        int sum = 0, num = n;
        while (num > 9) {
            sum = num;
            int s = 0;
            while (sum > 0) {
```

```
            s = s + (sum % 10);
            sum = sum / 10;
        }
        num = s;
    }
    if (num == 1) {
        System.out.println(n + " is a Magic Number.");
    } else {
        System.out.println(n + " is not a Magic Number.");
    }
    ob.close();
  }
}
```

I. Errors in the code:

1. **Line 13:** The condition should be while (sum > 0) to process all digits.
2. **Line 14:** Correct logic should be s = s + (sum % 10) to accumulate the digit sum.
3. **Line 15:** It should be sum = sum / 10 to remove the last digit.

II. Breakpoints:

- **Line 12:** Verify digit processing.
- **Line 14:** Check if digit sum is updated correctly.
- **Line 19:** Ensure correct magic number result.

III. Steps to fix the errors:

1. Update condition on line 13 to while (sum > 0).
2. Change line 14 to s = s + (sum % 10).
3. Modify line 15 to sum = sum / 10.

IV. FIXED CODE:

```
import java.util.Scanner;


public class MagicNumberCheck {
   public static void main(String args[]) {
      Scanner ob = new Scanner(System.in);
```

```
        System.out.println("Enter the number to be checked.");
        int n = ob.nextInt();
        int num = n;
        int sum = 0;

        while (num > 9) {
            sum = num;
            int s = 0;

            while (sum > 0) {  // Corrected loop condition
                s = s + (sum % 10);  // Accumulate sum of digits
                sum = sum / 10;      // Remove the last digit
            }
            num = s;  // Update num with the sum of digits
        }

        if (num == 1) {
            System.out.println(n + " is a Magic Number.");
        } else {
            System.out.println(n + " is not a Magic Number.");
        }

        ob.close();
    }
}
```

---

5) Merge Sort:

Original Code:

```
// This program implements the merge sort algorithm for arrays of integers.
import java.util.*;
public class MergeSort {
```

```
public static void main(String[] args) {
   int[] list = {14, 32, 67, 76, 23, 41, 58, 85};
   System.out.println("before: " + Arrays.toString(list));
   mergeSort(list);
   System.out.println("after: " + Arrays.toString(list));
}

public static void mergeSort(int[] array) {
   if (array.length > 1) {
      int[] left = leftHalf(array + 1);
      int[] right = rightHalf(array - 1);

      mergeSort(left);
      mergeSort(right);

      merge(array, left, right);
   }
}

public static int[] leftHalf(int[] array) {
   int size1 = array.length / 2;
   int[] left = new int[size1];
   for (int i = 0; i < size1; i++) {
      left[i] = array[i];
   }
   return left;
}

public static int[] rightHalf(int[] array) {
   int size1 = array.length / 2;
   int size2 = array.length - size1;
```

```
        int[] right = new int[size2];
        for (int i = 0; i < size2; i++) {
            right[i] = array[i + size1];
        }
        return right;
    }

    public static void merge(int[] result, int[] left, int[] right) {
        int i1 = 0;
        int i2 = 0;

        for (int i = 0; i < result.length; i++) {
            if (i2 >= right.length || (i1 < left.length && left[i1] <= right[i2])) {
                result[i] = left[i1];
                i1++;
            } else {
                result[i] = right[i2];
                i2++;
            }
        }
    }
}
```

I. Errors in the code:

1. **Line 15:** Use leftHalf(array) instead of array + 1.
2. **Line 16:** Change rightHalf(array - 1) to rightHalf(array).
3. **Line 21:** Fix merge(array, left++, right--) to just merge(array, left, right).

II. Breakpoints:

- **Line 15:** Ensure left array is created properly.
- **Line 16:** Check right array generation.
- **Line 21:** Confirm merge operation works without modifying arrays.

III. Steps to fix the errors:

1. Correct calls on lines 15 and 16.
2. Update merge call on line 21.

IV. FIXED CODE:

```
import java.util.Arrays;


public class MergeSort {
   public static void main(String[] args) {
      int[] list = {14, 32, 67, 76, 23, 41, 58, 85};
      System.out.println("before: " + Arrays.toString(list));
      mergeSort(list);
      System.out.println("after: " + Arrays.toString(list));
   }

   public static void mergeSort(int[] array) {
      if (array.length > 1) {
         int[] left = leftHalf(array);
         int[] right = rightHalf(array);


         mergeSort(left);
         mergeSort(right);


         merge(array, left, right);
      }
   }


   public static int[] leftHalf(int[] array) {
      int size1 = array.length / 2;
      int[] left = new int[size1];
      for (int i = 0; i < size1; i++) {
         left[i] = array[i];
      }
```

```
        return left;
    }

    public static int[] rightHalf(int[] array) {
        int size1 = array.length / 2;
        int size2 = array.length - size1;
        int[] right = new int[size2];
        for (int i = 0; i < size2; i++) {
            right[i] = array[i + size1];
        }
        return right;
    }

    public static void merge(int[] result, int[] left, int[] right) {
        int i1 = 0;
        int i2 = 0;

        for (int i = 0; i < result.length; i++) {
            if (i2 >= right.length || (i1 < left.length && left[i1] <= right[i2])) {
                result[i] = left[i1];
                i1++;
            } else {
                result[i] = right[i2];
                i2++;
            }
        }
    }
}
```

---

6) Multiply Matrices:

Original Code:

```
// Java program to multiply two matrices
import java.util.Scanner;

class MatrixMultiplication {
   public static void main(String args[]) {
      int m, n, p, q, sum = 0, c, d, k;

      Scanner in = new Scanner(System.in);
      System.out.println("Enter the number of rows and columns of first matrix");
      m = in.nextInt();
      n = in.nextInt();

      int first[][] = new int[m][n];
      System.out.println("Enter the elements of first matrix");
      for (c = 0; c < m; c++)
         for (d = 0; d < n; d++)
            first[c][d] = in.nextInt();

      System.out.println("Enter the number of rows and columns of second matrix");
      p = in.nextInt();
      q = in.nextInt();

      if (n != p)
         System.out.println("Matrices with entered orders can't be multiplied with each other.");
      else {
         int second[][] = new int[p][q];
         int multiply[][] = new int[m][q];
         System.out.println("Enter the elements of second matrix");
         for (c = 0; c < p; c++)
            for (d = 0; d < q; d++)
               second[c][d] = in.nextInt();
```

```
        for (c = 0; c < m; c++) {
          for (d = 0; d < q; d++) {
            for (k = 0; k < p; k++) {
              sum = sum + first[c][k] * second[k][d];  // Corrected index usage
            }
            multiply[c][d] = sum;
            sum = 0;
          }
        }

        System.out.println("Product of entered matrices:-");
        for (c = 0; c < m; c++) {
          for (d = 0; d < q; d++)
            System.out.print(multiply[c][d] + " ");
          System.out.println();
        }
      }
      in.close();  // Close the scanner to avoid resource leak
    }
}
```

I. Errors in the code:

1. **Line 33:** Ensure correct index usage in sum = sum + first[c][k] * second[k][d];.
2. **Line 45:** Initialize sum at the beginning of the nested loop to prevent incorrect accumulation.
3. **Closing the Scanner:** Always close the scanner to avoid resource leaks.

II. Breakpoints:

- **Line 33:** Verify correct multiplication logic.
- **Line 45:** Check initialization of sum.
- **Final Output:** Confirm the product matrix is printed correctly.

III. Steps to fix the errors:

1. Confirm index correctness in multiplication.

2. Reset sum at the start of inner loop.
3. Ensure in.close() is present.

IV. FIXED CODE:

```
import java.util.Scanner;


class MatrixMultiplication {
   public static void main(String args[]) {
      int m, n, p, q;
      Scanner in = new Scanner(System.in);


      System.out.println("Enter the number of rows and columns of first matrix");
      m = in.nextInt();
      n = in.nextInt();


      int first[][] = new int[m][n];
      System.out.println("Enter the elements of first matrix");
      for (int c = 0; c < m; c++)
         for (int d = 0; d < n; d++)
            first[c][d] = in.nextInt();


      System.out.println("Enter the number of rows and columns of second matrix");
      p = in.nextInt();
      q = in.nextInt();


      if (n != p)
         System.out.println("Matrices with entered orders can't be multiplied with each other.");
      else {
         int second[][] = new int[p][q];
         int multiply[][] = new int[m][q];
         System.out.println("Enter the elements of second matrix");
         for (int c = 0; c < p; c++)
```

```
            for (int d = 0; d < q; d++)
                second[c][d] = in.nextInt();

        for (int c = 0; c < m; c++) {
            for (int d = 0; d < q; d++) {
                int sum = 0;  // Reset sum at the start of the loop
                for (int k = 0; k < p; k++) {
                    sum += first[c][k] * second[k][d];  // Ensure correct indexing
                }
                multiply[c][d] = sum;
            }
        }

        System.out.println("Product of entered matrices:-");
        for (int c = 0; c < m; c++) {
            for (int d = 0; d < q; d++)
                System.out.print(multiply[c][d] + " ");
            System.out.println();
        }
    }
    in.close();  // Close the scanner to avoid resource leak
  }
}
```

7. Quadratic Probing

Original Code:

```
import java.util.Scanner;

class QuadraticProbingHashTable {
  private int currentSize, maxSize;
  private String[] keys;
  private String[] vals;
```

```
public QuadraticProbingHashTable(int capacity) {
    currentSize = 0;
    maxSize = capacity;
    keys = new String[maxSize];
    vals = new String[maxSize];
}

public void makeEmpty() {
    currentSize = 0;
    keys = new String[maxSize];
    vals = new String[maxSize];
}

public int getSize() {
    return currentSize;
}

public boolean isFull() {
    return currentSize == maxSize;
}

public boolean isEmpty() {
    return getSize() == 0;
}

public boolean contains(String key) {
    return get(key) != null;
}

private int hash(String key) {
```

```
    return Math.abs(key.hashCode() % maxSize);
}

public void insert(String key, String val) {
    int tmp = hash(key);
    int i = tmp, h = 1;

    do {
        if (keys[i] == null) {
            keys[i] = key;
            vals[i] = val;
            currentSize++;
            return;
        }
        if (keys[i].equals(key)) {
            vals[i] = val;
            return;
        }
        i + = (i + h / h--) % maxSize;  // Error
    } while (i != tmp);
}

public String get(String key) {
    int i = hash(key), h = 1;
    while (keys[i] != null) {
        if (keys[i].equals(key)) {
            return vals[i];
        }
        i + = (i + h / h--) % maxSize;  // Error
    }
    return null;
```

```
    }

    public void remove(String key) {
        if (!contains(key)) return;
        int i = hash(key), h = 1;
        while (!key.equals(keys[i])) {
            i + = (i + h / h--) % maxSize;  // Error
        }
        keys[i] = vals[i] = null;
        for (i = (i + h / h--) % maxSize; keys[i] != null; i = (i + h / h--) % maxSize) {
            String tmp1 = keys[i], tmp2 = vals[i];
            keys[i] = vals[i] = null;
            currentSize--;
            insert(tmp1, tmp2);
        }
        currentSize--;
    }

    public void printHashTable() {
        System.out.println("\nHash Table: ");
        for (int i = 0; i < maxSize; i++) {
            if (keys[i] != null)
                System.out.println(keys[i] + " " + vals[i]);
        }
        System.out.println();
    }
}

public class QuadraticProbingHashTableTest {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
```

```
System.out.println("Hash Table Test\n\nEnter size:");
QuadraticProbingHashTable qpht = new QuadraticProbingHashTable(scan.nextInt());
char ch;

do {
   System.out.println("\nHash Table Operations\n");
   System.out.println("1. Insert");
   System.out.println("2. Remove");
   System.out.println("3. Get");
   System.out.println("4. Clear");
   System.out.println("5. Size");
   int choice = scan.nextInt();

   switch (choice) {
      case 1:
         System.out.println("Enter key and value:");
         qpht.insert(scan.next(), scan.next());
         break;
      case 2:
         System.out.println("Enter key:");
         qpht.remove(scan.next());
         break;
      case 3:
         System.out.println("Enter key:");
         System.out.println("Value = " + qpht.get(scan.next()));
         break;
      case 4:
         qpht.makeEmpty();
         System.out.println("Hash Table Cleared\n");
         break;
      case 5:
```

```
                System.out.println("Size = " + qpht.getSize());
                break;
            default:
                System.out.println("Wrong Entry\n");
                break;
        }

        qpht.printHashTable();
        System.out.println("\nDo you want to continue (Type y or n)\n");
        ch = scan.next().charAt(0);
      } while (ch == 'Y' || ch == 'y');
   }
}
```

Steps to Fix: I. Change i + = (i + h / h--) % maxSize; to i = (i + h * h++) % maxSize;.

II. Complete the comment for clarity.

III. Ensure proper functioning of the insert and get methods by fixing probing logic.

IV. Verify the remove method to ensure proper re-insertion of displaced keys.

Fixed Code:

```
import java.util.Scanner;

class QuadraticProbingHashTable {
   private int currentSize, maxSize;
   private String[] keys;
   private String[] vals;

   public QuadraticProbingHashTable(int capacity) {
      currentSize = 0;
      maxSize = capacity;
      keys = new String[maxSize];
      vals = new String[maxSize];
   }
```

```
public void makeEmpty() {
   currentSize = 0;
   keys = new String[maxSize];
   vals = new String[maxSize];
}

public int getSize() {
   return currentSize;
}

public boolean isFull() {
   return currentSize == maxSize;
}

public boolean isEmpty() {
   return getSize() == 0;
}

public boolean contains(String key) {
   return get(key) != null;
}

private int hash(String key) {
   return Math.abs(key.hashCode() % maxSize);
}

public void insert(String key, String val) {
   int tmp = hash(key);
   int i = tmp, h = 1;
```

```
    do {
        if (keys[i] == null) {
            keys[i] = key;
            vals[i] = val;
            currentSize++;
            return;
        }
        if (keys[i].equals(key)) {
            vals[i] = val;
            return;
        }
        i = (i + h * h++) % maxSize;  // Fixed quadratic probing logic
    } while (i != tmp);
}

public String get(String key) {
    int i = hash(key), h = 1;
    while (keys[i] != null) {
        if (keys[i].equals(key)) {
            return vals[i];
        }
        i = (i + h * h++) % maxSize;
    }
    return null;
}

public void remove(String key) {
    if (!contains(key)) return;
    int i = hash(key), h = 1;
    while (!key.equals(keys[i])) {
        i = (i + h * h++) % maxSize;
```

```
        }
        keys[i] = vals[i] = null;
        for (i = (i + h * h++) % maxSize; keys[i] != null; i = (i + h * h++) % maxSize) {
            String tmp1 = keys[i], tmp2 = vals[i];
            keys[i] = vals[i] = null;
            currentSize--;
            insert(tmp1, tmp2);
        }
        currentSize--;
    }

    public void printHashTable() {
        System.out.println("\nHash Table: ");
        for (int i = 0; i < maxSize; i++) {
            if (keys[i] != null)
                System.out.println(keys[i] + " " + vals[i]);
        }
        System.out.println();
    }
}

public class QuadraticProbingHashTableTest {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Hash Table Test\n\nEnter size:");
        QuadraticProbingHashTable qpht = new QuadraticProbingHashTable(scan.nextInt());
        char ch;

        do {
            System.out.println("\nHash Table Operations\n");
            System.out.println("1. Insert");
```

```
System.out.println("2. Remove");
System.out.println("3. Get");
System.out.println("4. Clear");
System.out.println("5. Size");
int choice = scan.nextInt();

switch (choice) {
   case 1:
      System.out.println("Enter key and value:");
      qpht.insert(scan.next(), scan.next());
      break;
   case 2:
      System.out.println("Enter key:");
      qpht.remove(scan.next());
      break;
   case 3:
      System.out.println("Enter key:");
      System.out.println("Value = " + qpht.get(scan.next()));
      break;
   case 4:
      qpht.makeEmpty();
      System.out.println("Hash Table Cleared\n");
      break;
   case 5:
      System.out.println("Size = " + qpht.getSize());
      break;
   default:
      System.out.println("Wrong Entry\n");
      break;
}
```

```
            qpht.printHashTable();
            System.out.println("\nDo you want to continue (Type y or n)\n");
            ch = scan.next().charAt(0);
        } while (ch == 'Y' || ch == 'y');
    }
}
```

8. Sorting Array

Original Code:

```
import java.util.Scanner;

public class Sorting Array {
    public static void main(String[] args) {
        int n;
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of elements: ");
        n = sc.nextInt();
        int[] arr = new int[n];
        System.out.print("Enter elements: ");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        bubbleSort(arr);
        System.out.print("Sorted array: ");
        for (int i : arr) {
            System.out.print(i + " ");
        }
    }

    static void bubbleSort(int[] arr) {
        for (int i = 0; i < arr.length - 1; i++) {
```

```
        for (int j = 0; j < arr.length - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
```

Steps to Fix: I. Rename class from Sorting Array to SortingArray (remove the space).

II. Ensure the proper format of user prompts and outputs.

III. Verify that sorting logic is correct and follows the intended algorithm.

IV. Test with different input sizes for robustness.

Fixed Code:

```
import java.util.Scanner;

public class SortingArray {
    public static void main(String[] args) {
        int n;
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of elements: ");
        n = sc.nextInt();
        int[] arr = new int[n];
        System.out.print("Enter elements: ");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        bubbleSort(arr);
        System.out.print("Sorted array: ");
        for (int i : arr) {
```

```
        System.out.print(i + " ");
    }
  }

  static void bubbleSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
      for (int j = 0; j < arr.length - i - 1; j++) {
        if (arr[j] > arr[j + 1]) {
          int temp = arr[j];
          arr[j] = arr[j + 1];
          arr[j + 1] = temp;
        }
      }
    }
  }
}
```

9. Stack Implementation

Original Code:

```
class Stack {
  private int maxSize;
  private int top;
  private int[] stackArray;

  public Stack(int size) {
    maxSize = size;
    stackArray = new int[maxSize];
    top = -1;
  }

  public void push(int value) {
    if (top == maxSize - 1) {
```

```
        System.out.println("Stack is full");
    } else {
        stackArray[++top] = value;
    }
}

public int pop() {
    if (top == -1) {
        System.out.println("Stack is empty");
        return -1; // Error code
    } else {
        return stackArray[top--];
    }
}

public int peek() {
    if (top == -1) {
        System.out.println("Stack is empty");
        return -1; // Error code
    } else {
        return stackArray[top];
    }
}

public boolean isEmpty() {
    return top == -1;
}

public boolean isFull() {
    return top == maxSize - 1;
}
```

```
    public void printStack() {
        if (isEmpty()) {
            System.out.println("Stack is empty");
        } else {
            System.out.print("Stack: ");
            for (int i = 0; i <= top; i++) {
                System.out.print(stackArray[i] + " ");
            }
            System.out.println();
        }
    }
}

public class StackTest {
    public static void main(String[] args) {
        Stack stack = new Stack(5);
        stack.push(10);
        stack.push(20);
        stack.push(30);
        stack.printStack();
        System.out.println("Top element is: " + stack.peek());
        stack.pop();
        stack.printStack();
        System.out.println("Is stack empty? " + stack.isEmpty());
        stack.pop();
        stack.pop();
        stack.pop(); // Extra pop to show empty condition
        stack.printStack();
    }
}
```

Steps to Fix: I. Ensure that the error messages are clear and indicate the operation being attempted.

II. Confirm that the stack's maximum size is correctly handled during push operations.

III. Validate the return values for pop and peek when the stack is empty.

IV. Enhance the printStack method for better visibility of the stack's content.

Fixed Code:

```
class Stack {
    private int maxSize;
    private int top;
    private int[] stackArray;

    public Stack(int size) {
        maxSize = size;
        stackArray = new int[maxSize];
        top = -1;
    }

    public void push(int value) {
        if (isFull()) {
            System.out.println("Cannot push " + value + ": Stack is full");
        } else {
            stackArray[++top] = value;
            System.out.println("Pushed: " + value);
        }
    }

    public int pop() {
        if (isEmpty()) {
            System.out.println("Cannot pop: Stack is empty");
            return -1; // Error code
        } else {
            return stackArray[top--];
```

```
    }
}

public int peek() {
    if (isEmpty()) {
        System.out.println("Cannot peek: Stack is empty");
        return -1; // Error code
    } else {
        return stackArray[top];
    }
}

public boolean isEmpty() {
    return top == -1;
}

public boolean isFull() {
    return top == maxSize - 1;
}

public void printStack() {
    if (isEmpty()) {
        System.out.println("Stack is empty");
    } else {
        System.out.print("Stack: ");
        for (int i = 0; i <= top; i++) {
            System.out.print(stackArray[i] + " ");
        }
        System.out.println();
    }
}
```

```
}

public class StackTest {
    public static void main(String[] args) {
        Stack stack = new Stack(5);
        stack.push(10);
        stack.push(20);
        stack.push(30);
        stack.printStack();
        System.out.println("Top element is: " + stack.peek());
        stack.pop();
        stack.printStack();
        System.out.println("Is stack empty? " + stack.isEmpty());
        stack.pop();
        stack.pop();
        stack.pop(); // Extra pop to show empty condition
        stack.printStack();
    }
}
```

---

10. Tower of Hanoi

Original Code:

```
public class TowerOfHanoi {
    public static void hanoi(int n, char from, char to, char aux) {
        if (n == 1) {
            System.out.println("Move disk 1 from " + from + " to " + to);
            return;
        }
        hanoi(n, from, aux, to);  // Error: should decrement n
        System.out.println("Move disk " + n + " from " + from + " to " + to);
```

```
        hanoi(n, aux, to, from);  // Error: should decrement n
    }

    public static void main(String[] args) {
        int n = 4; // Number of disks
        hanoi(n, 'A', 'C', 'B'); // A, B and C are names of rods
    }
}
```

Steps to Fix: I. Change hanoi(n, from, aux, to); to hanoi(n - 1, from, aux, to);.

II. Change hanoi(n, aux, to, from); to hanoi(n - 1, aux, to, from);.

III. Ensure that the recursive calls correctly handle the movement of disks.

IV. Test with different values of n to verify correctness.

Fixed Code:

```
public class TowerOfHanoi {
    public static void hanoi(int n, char from, char to, char aux) {
        if (n == 1) {
            System.out.println("Move disk 1 from " + from + " to " + to);
            return;
        }
        hanoi(n - 1, from, aux, to);  // Fixed: decrement n
        System.out.println("Move disk " + n + " from " + from + " to " + to);
        hanoi(n - 1, aux, to, from);  // Fixed: decrement n
    }

    public static void main(String[] args) {
        int n = 4; // Number of disks
        hanoi(n, 'A', 'C', 'B'); // A, B and C are names of rods
    }
}
```