

Experiment – 10

1. Store a list of integers in an `ArrayList<Integer>` using autoboxing. Iterate through the list, unbox each value, and determine if it is a prime number, printing the results.

Code :-

```
import java.util.ArrayList;
import java.util.List;

/**
 * Lab Experiment 10: Task 1
 * Demonstrates storing integers in an ArrayList using autoboxing,
 * iterating, unboxing, and checking for prime numbers.
 */
public class PrimeChecker {

    /**
     * Checks if a given number is prime.
     * @param number The integer to check.
     * @return true if the number is prime, false otherwise.
     */
    public static boolean isPrime(int number) {
        // Numbers less than 2 are not prime
        if (number < 2) {
            return false;
        }
        // Check for divisibility from 2 up to the square root of the number
        // (Optimization: no need to check beyond the square root)
        for (int i = 2; i * i <= number; i++) {
            if (number % i == 0) {
                return false; // Found a divisor, not prime
            }
        }
        return true; // No divisors found, it's prime
    }

    public static void main(String[] args) {
        // 1. Store a list of integers using autoboxing
        // Java automatically converts int primitives to Integer objects (autoboxing)
        List<Integer> numbers = new ArrayList<>();
        numbers.add(2);
```

```

numbers.add(3);
numbers.add(4);
numbers.add(5);
numbers.add(17);
numbers.add(18);
numbers.add(19);
numbers.add(23);
numbers.add(25);
numbers.add(1);
numbers.add(0);

System.out.println("Checking numbers in the list for primality:");
System.out.println("-----");

// 2. Iterate through the list
for (Integer numObject : numbers) {
    // 3. Unbox each value and determine if it's prime
    // Java automatically converts Integer object back to int primitive (unboxing)
    // when needed for calculations or method calls like isPrime(int).
    int numValue = numObject; // Explicit unboxing: int numValue = numObject.intValue();
    boolean prime = isPrime(numValue); // numObject is auto-unboxed here

    // 4. Print the results
    System.out.println(numValue + (prime ? " is prime." : " is not prime."));
}
System.out.println("-----");
}
}

```

```

C:\Users\arinb\.jdk\java17\bin\java.exe "-javaagent:C:\Users\arinb\AppData\
Checking numbers in the list for primality:
-----
2 is prime.
3 is prime.
4 is not prime.
5 is prime.
17 is prime.
18 is not prime.
19 is prime.
23 is prime.
25 is not prime.
1 is not prime.
0 is not prime.
-----
Process finished with exit code 0

```

2. Create an ArrayList to store Employee objects with attributes like name, id, and salary. Add three employees, update one employee's salary, remove another by their id, and print the remaining employees.

Code for Employee.java:-

```
/**
 * Represents an Employee with id, name, and salary.
 * Used in Lab Experiment 10, Task 2.
 */
public class Employee {
    private int id;
    private String name;
    private double salary;

    // Constructor
    public Employee(int id, String name, double salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }

    // Getters
    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public double getSalary() {
        return salary;
    }

    // Setter for salary (needed for update)
    public void setSalary(double salary) {
        this.salary = salary;
    }

    // toString() method for easy printing of Employee objects
    @Override
    public String toString() {
```

```

        return "Employee{" +
            "id=" + id +
            ", name=" + name + "\" +
            ", salary=" + String.format("%.2f", salary) + // Format salary
            "'}";
    }
}

```

Code for EmployeeManager.java:-

```

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

/**
 * Lab Experiment 10: Task 2
 * Manages a list of Employee objects using an ArrayList.
 * Demonstrates adding, updating, removing, and printing employees.
 */
public class EmployeeManager {

    public static void main(String[] args) {
        // 1. Create an ArrayList to store Employee objects
        List<Employee> employeeList = new ArrayList<>();

        // 2. Add three employees
        employeeList.add(new Employee(101, "Alice", 60000.00));
        employeeList.add(new Employee(102, "Bob", 75000.50));
        employeeList.add(new Employee(103, "Charlie", 55000.75));

        System.out.println("Initial Employee List:");
        printEmployeeList(employeeList);

        // 3. Update one employee's salary (e.g., Bob's salary)
        int employeeIdToUpdate = 102;
        double newSalary = 80000.00;
        boolean updated = false;
        for (Employee emp : employeeList) {
            if (emp.getId() == employeeIdToUpdate) {
                emp.setSalary(newSalary);
                System.out.println("\nUpdated salary for Employee ID: " + employeeIdToUpdate);
                updated = true;
                break; // Exit loop once updated
            }
        }
        if (!updated) {
            System.out.println("\nEmployee ID " + employeeIdToUpdate + " not found for update.");
        }
    }
}

```

```

// 4. Remove another employee by their id (e.g., Charlie)
int employeeIdToRemove = 103;
boolean removed = false;
// Using Iterator is the safest way to remove while iterating
Iterator<Employee> iterator = employeeList.iterator();
while (iterator.hasNext()) {
    Employee emp = iterator.next();
    if (emp.getId() == employeeIdToRemove) {
        iterator.remove(); // Safely remove the current element
        System.out.println("Removed Employee ID: " + employeeIdToRemove);
        removed = true;
        break; // Exit loop once removed
    }
}
if (!removed) {
    System.out.println("Employee ID " + employeeIdToRemove + " not found for removal.");
}

// 5. Print the remaining employees
System.out.println("\nFinal Employee List:");
printEmployeeList(employeeList);
}

/**
 * Helper method to print the list of employees.
 * @param list The list of employees to print.
 */
public static void printEmployeeList(List<Employee> list) {
    if (list.isEmpty()) {
        System.out.println(" List is empty.");
    } else {
        for (Employee emp : list) {
            System.out.println(" " + emp); // Uses the toString() method of Employee
        }
    }
    System.out.println("-----");
}
}

```

```

C:\Users\arinb\jdk\java17\bin\java.exe "-javaagent:C:\Users\arinb\AppData\Local\Programs\
Initial Employee List:
    Employee{id=101, name='Alice', salary=60000.00}
    Employee{id=102, name='Bob', salary=75000.50}
    Employee{id=103, name='Charlie', salary=55000.75}
-----

Updated salary for Employee ID: 102
Removed Employee ID: 103

Final Employee List:
    Employee{id=101, name='Alice', salary=60000.00}
    Employee{id=102, name='Bob', salary=80000.00}
-----

Process finished with exit code 0

```

3. Use a HashMap to manage a product inventory where keys are productId (Integer) and values are quantity (Integer). Add three products, update the quantity of one product, remove another, and display the final inventory.

Code:-

```
import java.util.HashMap;
import java.util.Map;

/**
 * Lab Experiment 10: Task 3
 * Manages a product inventory using a HashMap.
 * Demonstrates adding, updating, removing products and displaying the inventory.
 */
public class ProductInventory {

    public static void main(String[] args) {
        // 1. Use a HashMap to manage inventory (productId -> quantity)
        Map<Integer, Integer> inventory = new HashMap<>();

        // 2. Add three products
        inventory.put(1001, 50); // Product ID 1001, Quantity 50
        inventory.put(1002, 120); // Product ID 1002, Quantity 120
        inventory.put(1003, 75); // Product ID 1003, Quantity 75
        System.out.println("Initial Inventory:");
        printInventory(inventory);

        // 3. Update the quantity of one product (e.g., product 1001)
        int productIdToUpdate = 1001;
        int newQuantity = 45; // Decreased quantity
        if (inventory.containsKey(productIdToUpdate)) {
            inventory.put(productIdToUpdate, newQuantity); // put() replaces the value if key exists
            System.out.println("\nUpdated quantity for Product ID: " + productIdToUpdate + " to " +
newQuantity);
        } else {
            System.out.println("\nProduct ID " + productIdToUpdate + " not found for update.");
        }

        // 4. Remove another product (e.g., product 1002)
        int productIdToRemove = 1002;
        if (inventory.containsKey(productIdToRemove)) {
            inventory.remove(productIdToRemove);
        }
    }
}
```

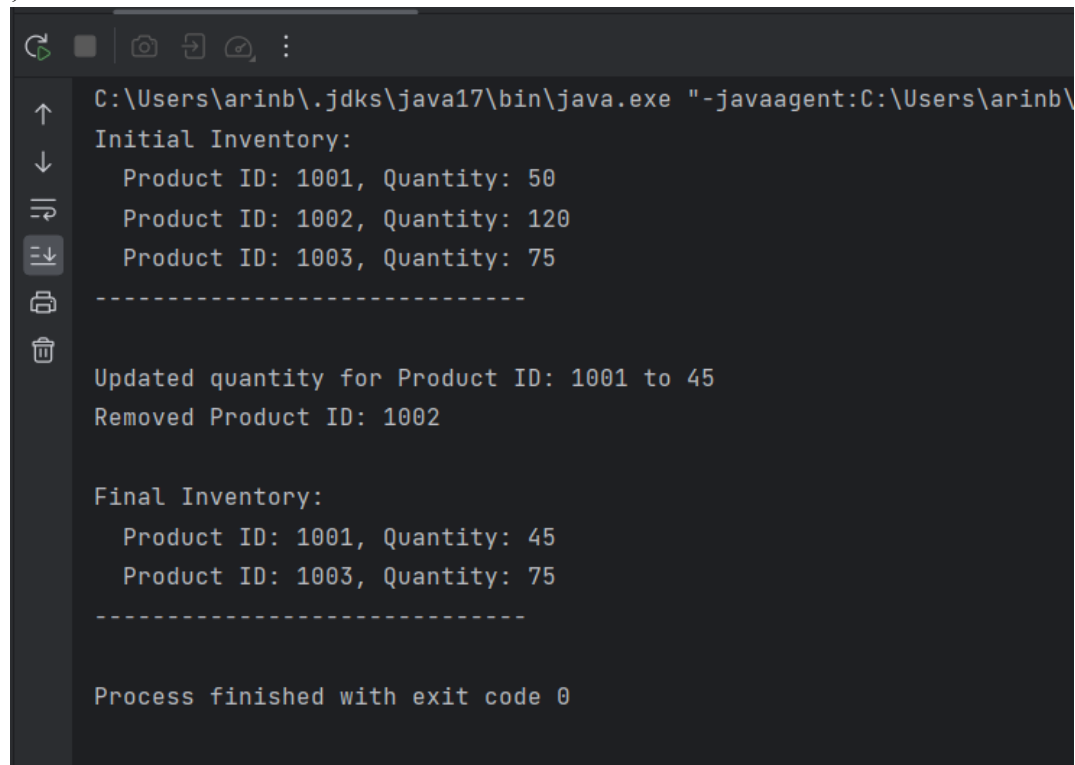
```

        System.out.println("Removed Product ID: " + productIdToRemove);
    } else {
        System.out.println("Product ID " + productIdToRemove + " not found for removal.");
    }
}

// 5. Display the final inventory
System.out.println("\nFinal Inventory:");
printInventory(inventory);
}

/**
 * Helper method to print the inventory map.
 * @param inventory The inventory map (productId -> quantity) to print.
 */
public static void printInventory(Map<Integer, Integer> inventory) {
    if (inventory.isEmpty()) {
        System.out.println(" Inventory is empty.");
    } else {
        // Iterate through the entry set for easy access to key and value
        for (Map.Entry<Integer, Integer> entry : inventory.entrySet()) {
            System.out.println(" Product ID: " + entry.getKey() + ", Quantity: " + entry.getValue());
        }
    }
    System.out.println("-----");
}
}
}

```



```

C:\Users\arinb\.jdk\java17\bin\java.exe "-javaagent:C:\Users\arinb\
Initial Inventory:
    Product ID: 1001, Quantity: 50
    Product ID: 1002, Quantity: 120
    Product ID: 1003, Quantity: 75
-----
Updated quantity for Product ID: 1001 to 45
Removed Product ID: 1002

Final Inventory:
    Product ID: 1001, Quantity: 45
    Product ID: 1003, Quantity: 75
-----

Process finished with exit code 0

```

4. Given an array of names with duplicates (e.g., ["Aman", "Varchasv", "Divyansh", "Varchasv", "Aman"]), store them in a HashSet to eliminate duplicates. Check if a specific name exists in the set and print the unique names.

Code:-

```
import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;

/**
 * Lab Experiment 10: Task 4
 * Demonstrates using a HashSet to store unique names from an array with duplicates.
 * Also checks for the existence of a specific name.
 */
public class UniqueNames {

    public static void main(String[] args) {
        // 1. Given array of names with duplicates
        String[] namesArray = {"Aman", "Varchasv", "Divyansh", "Varchasv", "Aman", "Sneha"};
        System.out.println("Original Array of Names: " + Arrays.toString(namesArray));

        // 2. Store them in a HashSet to eliminate duplicates
        // We can directly initialize the HashSet from the array using Arrays.asList()
        Set<String> uniqueNames = new HashSet<>(Arrays.asList(namesArray));

        // 3. Print the unique names
        System.out.println("\nUnique Names (from HashSet):");
        // Printing the set automatically shows only unique elements
        System.out.println(" " + uniqueNames);
        System.out.println("-----");

        // 4. Check if a specific name exists in the set
        String nameToCheck = "Divyansh";
        boolean exists = uniqueNames.contains(nameToCheck);
        System.out.println("\nChecking if \"" + nameToCheck + "\" exists in the set: " + exists);

        nameToCheck = "Rahul";
        exists = uniqueNames.contains(nameToCheck);
        System.out.println("Checking if \"" + nameToCheck + "\" exists in the set: " + exists);
        System.out.println("-----");
    }
}
```



```
}
```

```
C:\Users\arinb\.jdk\java17\bin\java.exe "-javaagent:C:\Users\arinb\AppData\l
Original Array of Names: [Aman, Varchasv, Divyansh, Varchasv, Aman, Sneha]

Unique Names (from HashSet):
    [Varchasv, Aman, Sneha, Divyansh]
-----

Checking if "Divyansh" exists in the set: true
Checking if "Rahul" exists in the set: false
-----

Process finished with exit code 0
```

5. Given an ArrayList of integers (possibly with duplicates), find the sum of its unique values. [Hint: Convert it to HashSet]

Code:-

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

/**
 * Lab Experiment 10: Task 5
 * Finds the sum of unique values from an ArrayList containing duplicates,
 * using a HashSet to filter unique elements.
 */
public class SumUniqueValues {

    public static void main(String[] args) {
        // 1. Given an ArrayList of integers (possibly with duplicates)
        // Example Input: [3, 5, 3, 8, 2, 5]
        List<Integer> numbersWithDuplicates = new ArrayList<>(Arrays.asList(3, 5, 3, 8, 2, 5, 8, 10));
        System.out.println("Original List with Duplicates: " + numbersWithDuplicates);

        // 2. Convert it to HashSet to get unique values
```

```

// The HashSet constructor automatically handles duplicate removal
Set<Integer> uniqueNumbers = new HashSet<>(numbersWithDuplicates);
System.out.println("Unique Values (from HashSet): " + uniqueNumbers);

// 3. Find the sum of its unique values
int sum = 0;
// Iterate through the unique values in the HashSet
for (Integer uniqueValue : uniqueNumbers) {
    sum += uniqueValue; // Auto-unboxing happens here
}

// 4. Print the sum
System.out.println("Sum of Unique Values: " + sum);
System.out.println("-----");
}
}

```

```

C:\Users\arinb\.jdk\java17\bin\java.exe "-javaagent:C:\Users\
Original List with Duplicates: [3, 5, 3, 8, 2, 5, 8, 10]
Unique Values (from HashSet): [2, 3, 5, 8, 10]
Sum of Unique Values: 28
-----

Process finished with exit code 0

```