National University of Computer and Emerging Sciences

Laboratory Manual

for

Data Structures Lab

Course Instructor	Sir Waqas Ali
Lab Instructor(s)	Zain Ali Nasir, Hira Tayyab
Section	BSE 5C
Date	Wednesday, Sep 11, 2024
Semester	Fall 2024

Department of Software Engineering
FAST-NU, Lahore, Pakistan

In this lab we cover:

- Generics and Collections

What are Generics in Java?
Generics in Java allow for type-safe code by enabling classes, methods, and interfaces to operate on any specified data type without needing to cast or use `Object` type. The key concept behind generics is type parameterization, which means you can define a class or method that can work with any type, while still ensuring type safety.

Key Benefits of Generics:

1. **Type Safety:** Generics enforce compile-time type checking, reducing runtime errors like `ClassCastException`. For example, using a `List<Integer>` ensures that only integers can be added.
2. **Reusability:** You can write a single generic method or class that works with different types, eliminating the need for code duplication.
3. **Elimination of Casting:** Since you specify the type, you avoid unnecessary casting of objects

Example:

```
// Generic class

class Box<T> {

   private T item;

   public void add(T item) {

       this.item = item;

   }

   public T get() {

       return item;

   }

}
```

```java
public class Main {

   public static void main(String[] args) {

       //Same class behave as a integer

       Box<Integer> integerBox = new Box<>();

       integerBox.add(100);

       Integer value = integerBox.get(); // No need for casting

       System.out.println(value);

       //Same class behave as an String

       Box<String> str = new Box<>();

       str.add("This is good.");

       String value1 = str.get(); // No need for casting

       System.out.println(value1);

   }

}
```

What are Collections?

Collections in Java are a set of classes and interfaces that provide data structures like lists, sets, and maps to store and manipulate groups of objects. The Java Collections Framework (JCF) is part of the `java.util` package and offers various implementations of these data structures.

Key Interfaces of Java Collections:

1. **List:** An ordered collection (also known as a sequence) that allows duplicate elements. Implementations include `ArrayList`, `LinkedList`, etc.
2. **Set:** A collection that does not allow duplicate elements. Implementations include `HashSet`, `TreeSet`, etc.
3. **Map:** A collection that stores key-value pairs, where keys are unique.

```java
import java.util.ArrayList;

import java.util.List;

public class CollectionExample {

   public static void main(String[] args) {

       // Create a List of Strings

       List<String> names = new ArrayList<>();

       names.add("BS-SE(5A)");

       names.add("BS-SE(6A)");

       names.add("BS-SE(7A)");



       // Iterate through the List

       for (String name : names) {

           System.out.println(name);

       }

   }

}
```

List Collections

```
List <data-type> list1= new ArrayList();
List <data-type> list2 = new LinkedList();
List <data-type> list3 = new Vector();
List <data-type> list4 = new Stack();
```

Set Collections

```
Set<data-type> s1 = new HashSet<data-type>();
Set<data-type> s2 = new LinkedHashSet<data-type>();
Set<data-type> s3 = new TreeSet<data-type>();
```

Map Collections

Class	Description
HashMap	HashMap is the implementation of Map, but it doesn't maintain any order.
LinkedHashMap	LinkedHashMap is the implementation of Map. It inherits HashMap class. It maintains insertion order.
TreeMap	TreeMap is the implementation of Map and SortedMap. It maintains ascending order.

Code Example:

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


public class MapExample {
    public static void main(String[] args) {
        // Create a HashMap to store students' names and their grades
        Map<String, Integer> studentGrades = new HashMap<>();

        // Add key-value pairs to the map
        studentGrades.put("Ali", 85);
        studentGrades.put("Umer", 92);
        studentGrades.put("AbuBakar", 78);
        studentGrades.put("Usman", 90);

        // Print the map
        System.out.println("Initial Map: " + studentGrades);

        // Access value using a key
```

```
        int aliceGrade = studentGrades.get("Umer");
        System.out.println("Umer's Grade: " + aliceGrade);

        // Check if a key exists
        if (studentGrades.containsKey("AbuBakar")) {
            System.out.println("AbuBakar is in the map.");
        }

        // Iterate over the map's entries
        System.out.println("Student Grades:");
        for (Map.Entry<String, Integer> entry : studentGrades.entrySet()) {
            System.out.println(entry.getKey() + ": " + entry.getValue());
        }
        // Remove a key-value pair
        studentGrades.remove("Usman");
        System.out.println("After removing Usman: " + studentGrades);

        // Replace a value for an existing key
        studentGrades.put("AbuBakar", 95);
        System.out.println("After updating David's grade: " + studentGrades);
    }
}
```

Iterators

Iterator object, which allows a program to walk through the collection and remove elements from it during the iteration.

Code Example:

```
Iterator< String > iterator = collection1.iterator();
// loop while collection has items
while ( iterator.hasNext() )
{
if ( collection2.contains( iterator.next() ) )
iterator.remove(); // remove current Color
} // end while
```

Map Iteration

1) entrySet

```
Iterator<Map.Entry<Integer, String>> iterator = map.entrySet().iterator(); // Iterate using the Iterator
while (iterator.hasNext())
{
Map.Entry<Integer, String> entry = iterator.next();
System.out.println(entry.getKey() + " " + entry.getValue());
}
```

2) KeySet

```
Iterator<Integer> iterator = map.keySet().iterator();
while (iterator.hasNext())
 {
Integer key = iterator.next();
String value = map.get(key);
System.out.println(key + " " + value);
}
```

Lab Manual

Problem Statement 1: Remove Elements from a LinkedList

Write a Java program that creates and manages three types of `LinkedList` collections: one for `Integer`, one for `String`, and one for `Character`. The program should be console-based and provide the following options in a menu:

Console Menu:

1. Integer type LinkedList
2. String type LinkedList
3. Character type LinkedList
4. Exit

Functionality:

- Based on the user's choice (1, 2, or 3), the program should create a `LinkedList` of the specified type.
- The user will be prompted to enter multiple values for the selected `LinkedList` (e.g., integers if option 1 is selected).
- After the values are entered, the program will generate random indices and remove the elements at those indices from the list.
- The removed elements should be stored in a separate collection (for each type) and displayed.
- The remaining elements in the original `LinkedList` should also be displayed after removal.
- The menu will then be displayed again until the user chooses the "Exit" option.

Example:

Enter option here: 1

// Integer type LinkedList is created

Enter Data:

1

2

6

4

// Randomly generated 3 numbers between 0 and the size of LinkedList: (e.g., 0, 2, 3)

// Remove elements from LinkedList -> Remaining LinkedList: [2]

// Display removed elements -> [1, 6, 4]

// Display menu again

Repeat the process until the user chooses to exit.

Problem Statement 2: Search for a Word in Files and Display Word Frequencies

Write a Java function that performs the following tasks:

1. Read Files from a Directory: The program should read multiple text files from a directory specified by the user.

2. Search for a Word: The user will enter a word to search for in the files. The search should be case-sensitive.

3. Display Files with Word Occurrences:

- For each file, display the top 3 files that contain the searched word, based on the frequency of the word in each file.
- Do not display files where the frequency of the searched word is zero.

4. Display Word Frequencies:

- For each file, display the frequency of all words in the file (in the form of a `Map` collection where the word is the key and the frequency is the value).

5. Use a `Map` Collection: Store the word frequencies for each file using a `Map<String, Integer>` where the key is the word and the value is the frequency.

6. File Processing:

- Each word is delimited by spaces and punctuation marks.
- Ignore empty words.
- The search word frequency and word frequency of all words should be displayed in the output.

Example:

Directory:

The demo directory contains the following 3 files:

demo1.txt:

This is Pakistan. Pakistan is very beautiful.

demo2.txt:

Pakistan has a very rich culture. People in Pakistan are very cooperative.

demo3.txt:

Worth to live place, Pakistan.

User Input:

Search word: “Pakistan”

Output:

demo1.txt:

{ "This": 1, "is": 2, "Pakistan": 2, "very": 1, "beautiful": 1 }

Frequency of 'Pakistan': 2

demo2.txt:

{ "Pakistan": 2, "has": 1, "very": 2, "rich": 1, "culture": 1, "People": 1, "in": 1, "are": 1, "cooperative": 1 }

Frequency of 'Pakistan': 2

demo3.txt:

{ "Worth": 1, "to": 1, "live": 1, "place": 1, "Pakistan": 1 }

Frequency of 'Pakistan': 1

Top 3 files based on the frequency of the word 'Pakistan':

1. demo1.txt: 2 occurrences
2. demo2.txt: 2 occurrences
3. demo3.txt: 1 occurrence

Notes:

- **Files with a word frequency of zero should not be displayed in the output.**
- **The `Map<String, Integer>` for word frequency should be displayed for each file before showing the search word frequency.**