National University of Computer & Emerging Sciences, Karachi
Computer Science Department
Spring 2023, Lab Manual – 11

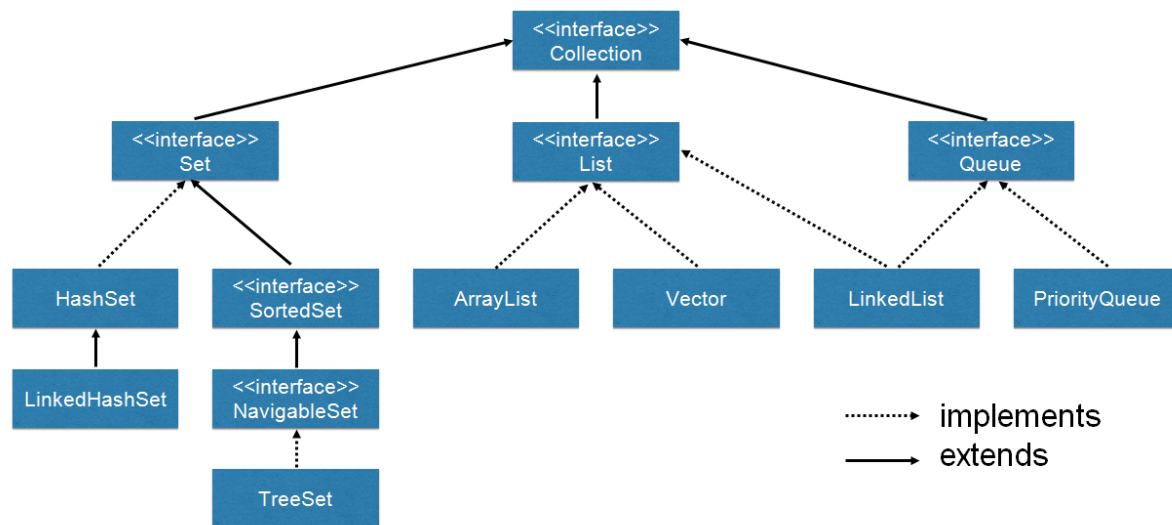| Course Code: CL-217	Course : Object Oriented Programming Lab
Instructor(s) :	Abeer Gauher, Hajra Ahmed, Shafique Rehman

LAB - 11

Generics in Java

File Handling in Java

Collection Framework:

The Java collections framework provides a set of interfaces and classes to implement various data structures and algorithms like Stacks, queues, arrays, linked list etc.

Generics:

Generic Classes:

Consider an example, let's say I want a class that prints an integer value. So, I write a piece of code like this.

```java
class PrintInteger{
    Integer i;
    public PrintInteger(Integer i) {
        this.i = i;
    }
    void print(){
        System.out.println(i);
    }
}
public class Mains {
    public static void main(String[] args) {
        PrintInteger p=new PrintInteger( i: 12);
        p.print();
```

Now, for printing a String object we have to replicate the code for String class and after that for other data types as well.

```java
class PrintInteger{
    Integer i;
    public PrintInteger(Integer i) {
        this.i = i;
    }
    void print(){
        System.out.println(i);
    }
}
class PrintString{
    String i;
    public PrintString(String i) {
        this.i = i;
    }
    void print(){
        System.out.println(i);
    }
}
public class Mains {
    public static void main(String[] args) {
        PrintInteger p=new PrintInteger( i: 12);
        p.print();
        PrintString p2=new PrintString( i: "HA");
        p2.print();
    }
}
```

So, it's better to make use of Generics concept in Java. Here we will make a generic class and use it for printing objects of different data types. The class name is followed by a type parameter section. The type parameter section of a generic class can have one or more type parameters separated by commas.

```java
class Printing<T>{
    T i;
    public Printing(T i) {
        this.i = i;
    }
    void print(){
        System.out.println(i);
    }
}
public class Mains {
    public static void main(String[] args) {
        Printing<Integer> p=new Printing<>( i: 23);
        p.print();
        Printing<String> s=new Printing<>( i: "HA");
        s.print();
    }
}
```

Generics are frequently used with collection frameworks in Java. For example,

```
ArrayList<Integer> number =new ArrayList<>();
```

Generics avoid the type Cast Exceptions as they are type safe.

For example, you might think that lets create an array list of Object type and add any type of Item to it. But then type casting error might occur. Shown in example below;

```
12  public class Mains {
13      public static void main(String[] args) {
14          ArrayList<Object> numbers =new ArrayList<>();
15          numbers.add(12);
16          numbers.add("Hajra");
17          numbers.add(12.5);
18
19          ArrayList<Integer> i=new ArrayList<>();
20          i.addAll(numbers);
21          System.out.println(i);
```

```
Build Output
C:\Users\ismail ahmed\IdeaProjects\lab 10\src\main\java\Mains.java:20:18
java: incompatible types: java.util.ArrayList<java.lang.Object> cannot be converted to java.util.Collection<? extends java.lang.Integer>
```

So, generics ensure type safety as well.

A generic class may have multiple parameters.

```
public class Box<T, S> {
    private T t;
    private S s;
    public void add(T t, S s) {
        this.t = t;
        this.s = s;
    }
    public T getFirst() {
        return t;
    }
    public S getSecond() {
        return s;
    }
    public static void main(String[] args) {
        Box<Integer, String> box = new Box<Integer, String>();
        box.add( t: 89, s: "HA");
        System.out.print(box.getFirst());
        System.out.print(box.getSecond());

        Box<String, String> box1 = new Box<String, String>();
        box1.add( t: "Message", s: "Hello World");
        System.out.print(box1.getFirst());
        System.out.print(box1.getSecond());
    }
}
```

Type Parameters

The common type parameters are as follows:

T - Type

E - Element

K - Key

N - Number

V - Value

Generic Methods:

We can also write generic functions that can be called with different types of arguments based on the type of arguments passed to the generic method. The compiler handles each method. Following are the rules to define Generic Methods :

- All generic method declarations have a type parameter section delimited by angle brackets (< and >) that precedes the method's return type (< T> in the example shown below).

- Each type parameter section contains one or more type parameters separated by commas. A type parameter, also known as a type variable, is an identifier that specifies a generic type name.

- The type parameters can be used to declare the return type and act as placeholders for the types of the arguments passed to the generic method, which are known as actual type arguments.

- Note that type parameters can represent only reference types, not primitive types

```
Mains.java      Box.java      mains3.java      GenericMethods.java
2
3 ▶    public class GenericMethods {
4          static < T > void Expressions(T t1){
5              System.out.println(t1+"!");
6          }
7 ▶        public static void main(String[] args) {
8              Expressions( t1: "Hurray");
9              Expressions( t1: "Alas");
10             Expressions( t1: "Wow");
11         }
12     }
13
```

Example 2:

```
public class Main
{
    public static < V > void printArray(V[] elements) {
        for ( V element : elements){
            System.out.print(element+" ");
        }
        System.out.println();
    }
    public static void main(String[] args) {
        Integer[] intArray = { 10, 20, 30, 40, 50 };
        Character[] charArray = { 'O', 'O', 'P', 'J','A','V','A'};

        System.out.println( "Printing Integer Array" );
        printArray( intArray  );

        System.out.println( "Printing Character Array" );
        printArray( charArray );
    }
}
```

```
Printing Integer Array
10 20 30 40 50
Printing Character Array
O O P J A V A
```

We can also use wildcards when we don't know about the parameter being passed to the methods like shown below. Here its not known wheter it'll be an integer or string or whatever:

```java
    public static <Lists> void showing(List<?> Lists){
        System.out.println(Lists);
    }
}
```

```java
    List<Integer> lists=new ArrayList<>();
    lists.add(20);
    lists.add(50);
    showing(lists);
    List<String> List2=new ArrayList<>();
    List2.add("H");
    List2.add("A");
    showing(List2);
```

Don'ts for generics:

- Generics don't work with primitive data types.
- Don't make type parameters as static in a generic class
- Don't instantiate a type parameter with in a class.
- Overloading is not supported in generic class

File Handling:

File handling in java comes under IO operations. Java IO package java.io classes are specially provided for file handling in java. We will discuss some operations on filing as:

- Create file
- Delete file
- Read file
- Write file
- Change file permissions

Different streams are used for file IO operations. ;ets discuss them first then we will proceed to IO operations.

Streams in Java:

In Java, a sequence of data is known as a stream. This concept is used to perform I/O operations on a file. There are two types of streams namely input Streams & output streams

Input Streams:

The Java InputStream class is the superclass of all input streams. The input stream is used to read data from numerous input devices like the keyboard, network, etc. We will use different methods of InputStream like read and close.

```java
InputStream obj = new FileInputStream(f1);
```
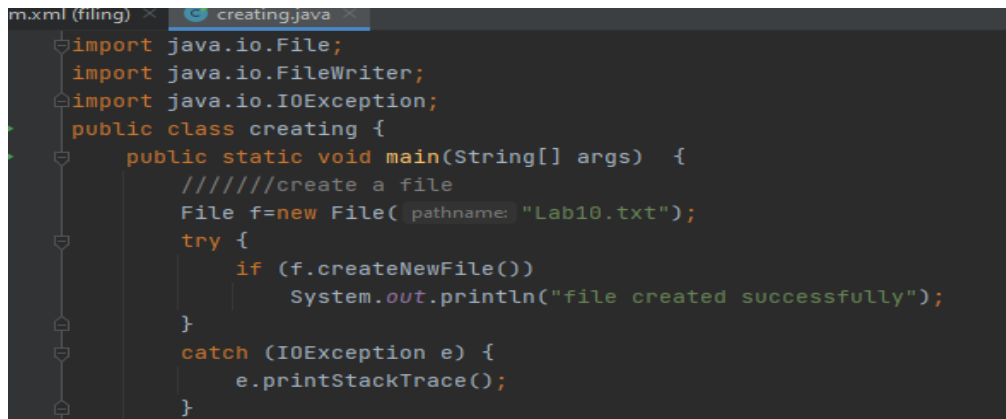
Output Streams:

The output stream is used to write data to numerous output devices like the monitor, file, etc. We will use different methods of OutputStream like Write and close.

```java
OutputStream obj2 = new FileOutputStream(f1);
```

Create a file:

We can use File class createNewFile() method to create new file. This method returns true if file is successfully created, otherwise it returns false.

```java
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
public class creating {
    public static void main(String[] args)  {
        ///////create a file
        File f=new File( pathname: "Lab10.txt");
        try {
            if (f.createNewFile())
                System.out.println("file created successfully");
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Delete a file:

We can use File class delete() method to remove file.

```java
m.xml (filing)  ×    creating.java  ×
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
public class creating {
    public static void main(String[] args)  {
        ///////delete a file
        File f=new File( pathname: "Lab10.txt");
        boolean delete = f.delete();
        if(delete)
            System.out.println("file deleted");
        else
            System.out.println("file not deleted");
```

Write a file:

We have four options for writing to a file in Java.

- FileWriter
- BufferedWriter (uses internal buffer to write data, used when you have more write operations)
- FileOutputStream (use for writing raw stream data to be written into file)
- Files (Internally, it's using OutputStream to write byte array into file).

We will uae FileWriter as FileWriter is the simplest way to write a file in Java. It provides overloaded write method to write int, byte array, and String to the File. You can also write part of the String or byte array using FileWriter. FileWriter writes directly into Files and should be used only when the number of writes is less.

```java
File f1=new File( pathname: "filing in java.txt");
FileWriter fileWriter=null;
String data="Hello, we are learning file handling in Java";
try{
    fileWriter=new FileWriter(f1);
    fileWriter.write(data);
    System.out.println("done");
}
catch (IOException e){
    e.printStackTrace();
}
```

Writing using BufferedWriter

```java
//create a file
File files=new File( pathname: "newjavafile.txt");
files.createNewFile();
//create a Buffered writer Stream
BufferedWriter bw=new BufferedWriter(new FileWriter(files));
//write to a file
bw.write( str: "Mr. H. Potter");
bw.write( str: "4, Privet Drive");
bw.write( str: "\nLittle Winging Street,surrey");
//close the file
bw.close();
```

Read a file:

We use BufferedWriter to read file and print to console.

```java
//create input stream
BufferedReader br=new BufferedReader(new FileReader( fileName: "newjavafile.txt"));
//to read the text till end of file
String buffer;
while((buffer = br.readLine())!= null){
    System.out.println(buffer);
}
    br.close();
}
```

Changing Permissions of a file:

To set and check the status for the Read Write & eXecutable permissions we can do:

```java
File f=new File( pathname: "abc.txt");
f.createNewFile();

f.setReadable(true);
f.setExecutable(true);
f.setWritable(true);

System.out.println(f.canRead());
System.out.println(f.canExecute());
System.out.println(f.canWrite());
}
```

Lab Tasks

Task 1:

Create an arraylist of 5 elements added by user (The user should input the elements). Then write this list to the text file present on the Desktop of your PC using BufferedWriter. After writing, read from the file and show the items in the list. Then,delete the file.

Task 2:

1. Create a text file for a student. You can use the student's name as the name of the file.
2. Write the student's full name and student ID in the file.
3. Then create an arraylist that represents the courses that the student has registered for the semester. Before registering for OOP Lab and OOP Theory check if the student has cleared both PF Lab and Theory. If the student has obtained 50 or above 50 in PF Lab and Theory then add OOP Lab and OOP theory in the arraylist.
4. Each course is assumed to be of 3 credit hours so student can register for more than 15 hours in the semester. Be careful while adding the courses in the list.
5. Write all this information to the file. Read from the file and then display all the content.

Task 3:

Write a generic method implementation to find the maximum element in an array of any type.

Hint: The max method takes an array of type T, which must implement the Comparable<T> interface. This interface provides the compareTo method, which allows us to compare two elements of type T.

```
public static <T extends Comparable<T>> int compare(T t1, T t2){
            return t1.compareTo(t2);
      }
```

Task 4:

Assuming you have a List of objects of different types such as Integer and Double, write a program in Java using generics that:
1. Sorts the list in ascending order based on the values of the objects.
2. Calculates the sum of all the objects in the list.
3. Finds the maximum value of all the objects in the list.
4. Prints out the sorted list, sum of all the objects, and maximum value of all the objects.
You'll need to use Java's Comparable and Wildcard generics to accomplish this.

Task 5:

Create a text file named as "Confidential.txt". Now, take input of designation from user. If the entered input is "Faculty", then give the RWX permissions to user. If the designation is student then give only the read permissions.

After assigning the permissions, show the Access Rights' status.