CSE1007-JAVA PROGRAMMING-LAB

EXERCISE-03

Name: Kulvir Singh

Reg. No.: 19BCE2074

Question 1:

Implement the following scenario using Java

There are four classes:

- Buffer, Producer, Consumer, and ProducerConsumerTest
- An object of the Buffer class has an integer data element that will be produced by the producer and consumed by the consumer.
- We have to synchronize the access to the buffer, so the Producer produces a new data element only when the Buffer is empty and the Consumer consumes the buffer's data only when it is available.
- The ProducerConsumerTest class is used to test the program.

Aim:

To implement the four scenarios and test it accordingly

Algorithm:

1. Start
2. Create a consumer class that uses thread
3. Create a Buffer
4. Initialize the Buffer using a constructor
5. Create the run method for the thread
6. Similarly create the producer class
7. Create a synchronized method for proper functioning of the threads
8. Create a main method to invoke and execute the threads
9. Stop

Code:

```
import java.util.*;

class Consumer extends Thread {

  private Buffer buffer;

  public Consumer(Buffer buffer) {
```

```
    this.buffer = buffer;
  }
  public void run() {
    int data;
    while (true) {
      data = buffer.consume();
    }
  }
}
public class prod_con {
  public static void main(String[] args) {
    Buffer buffer = new Buffer();
    Producer p = new Producer(buffer);
    Consumer c = new Consumer(buffer);
    p.start();
    c.start();
  }
}
class Producer extends Thread {
  private Buffer buffer;
  public Producer(Buffer buffer) {
    this.buffer = buffer;
  }
  public void run() {
    Random r = new Random();
    while (true) {
      int n = r.nextInt();
      buffer.produce(n);
    }
```

```
  }
}
class Buffer {
  private int data;
  private boolean e;
  public Buffer() {
    this.e = true;
  }
  public synchronized void produce(int nData) {
    while (!this.e) {
      try {
        this.wait();
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
    this.data = nData;
    this.e = false;
    this.notify();
    System.out.println("Produced:" + nData);
  }
  public synchronized int consume() {
    while (this.e) {
      try {
        this.wait();
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
```

```
    this.e = true;
    this.notify();
    System.out.println("Consumed:" + data);
    return data;
  }
}
```

Output:

```
Produced:-1032661496
Consumed:-1032661496
Produced:-1726658334
Consumed:-1726658334
Produced:1569298724
Consumed:1569298724
Produced:1496065195
Consumed:1496065195
Produced:-1257405720
Consumed:-1257405720
Produced:-510209610
Consumed:-510209610
Produced:-426597574
Consumed:-426597574
Produced:821235624
Consumed:821235624
Produced:-229801223
Consumed:-229801223
Produced:558434722
Consumed:558434722
Produced:2135904061
Consumed:2135904061
Produced:-450008486
Consumed:-450008486
```

Question -2

Write a java program to count the number of lines in a file. Assume that the files are text files. If an error occurs while trying to read from the files, you should print an error message for that file.

Also perform the following:-

- Get the list of all file/directory name for the current location
- Check if a file or directory has read and write permission
- Get last modified time of a file and size of the file

Aim:

To count the number of lines in a file and display the list of all files along with file size and their ability to be readable or writeable present in a directory.

Algorithm:

1. Start
2. Create a class
3. Create a function to check the readability of a file
4. Create a function to check the writability of a file.
5. Create a function to display the last modified date of a file.
6. Create a function to display the size of a file.
7. Create the main method and use file reader to count the number of word occurrences in a file
8. Invoke the created function to display the required outputs
9. Stop

Code:

```
import java.io.*;

import java.util.*;

class filehandling{

   static String isReadable(String x){

      File f = new File(x);

      if(f.canRead())

      return "Readable";

      else

      return "Not Readable";

   }
```

```
static String isWriteable(String x){
   File f = new File(x);
   if(f.canWrite())
   return "Writeable";
   else
   return "Not Writeable";
}
static Date datemodified(String x){
   File f = new File(x);
   long d = f.lastModified();
   Date o = new Date(d);
   return o;
}
static long size(String x){
   File f = new File(x);
   long s = f.length();
   return s;
}
public static void main(String[] args)throws IOException{

   String file_name="C:/Users/kulvir/Desktop/JAVA CODes/files/testoutput.txt";

   FileReader fr = new FileReader(file_name);
   BufferedReader br = new BufferedReader(fr);

   int i=0;

   while(br.readLine()!=null)
   i++;
```

```
        br.close();
        fr.close();

        System.out.println("The number of lines int the testoutput.txt file is = "+i+"\n");

        //all files in directory

        String directory = "C:/Users/kulvir/Desktop/JAVA CODes/files";

        File f = new File(directory);

        String content[] = f.list();

        for(i=0;i<content.length;i++){
           System.out.println(content[i]);
           System.out.println(isReadable(content[i]));
           System.out.println(isWriteable(content[i]));
           System.out.println("Last modified on = "+datemodified(content[i]));
           System.out.println("File Size in BYTES = "+size(content[i])+"\n");
        }
     }

}
```

Output:

```
C:\Users\kulvir\Desktop\JAVA CODes\files>javac filehandling.java

C:\Users\kulvir\Desktop\JAVA CODes\files>java filehandling
The number of lines int the testoutput.txt file is = 6

BankDemo.class
Readable
Writeable
Last modified on = Tue Sep 08 21:59:03 IST 2020
File Size in BYTES = 1171

CheckingAccount.class
Readable
Writeable
Last modified on = Tue Sep 08 21:59:03 IST 2020
File Size in BYTES = 1061

CheckingAccount.java
Readable
Writeable
Last modified on = Tue Sep 08 21:58:59 IST 2020
File Size in BYTES = 942

count_file.class
Readable
Writeable
Last modified on = Tue Sep 08 19:10:29 IST 2020
File Size in BYTES = 2397

count_file.java
Readable
Writeable
Last modified on = Tue Sep 08 19:10:23 IST 2020
```

Question -3

Demonstrate a user defined exception named *InsufficientFundsException*. Define a class *CheckingAccount* which contains *withdraw()* method that throws an InsufficientFundsException. Another class BankDemo demonstrates the invoking of deposit() and withdraw() methods . One more class CheckingAccount will check the balance.

Aim:

To demonstrate the user defined exception name InsufficientFundsException using methods and exception handling.

Algorithm:

1. Start
2. Create a class
3. Create a method which withdraws and displays an exception
4. Create a method which deposits and checks the balance and displays proper message
5. Create a main method to display and execute the code
6. Stop

Code:

```
class InsufficientFundsException extends Exception{
   public InsufficientFundsException(String s){
      super(s);
   }
}
class BankDemo{
   static double bal=0.0;

   static void deposit(double x){
      bal+=x;
   }
```

```
    public static void main(String[] args) {
        CheckingAccount o = new CheckingAccount();
        System.out.println("Current Balance = "+bal);
        o.withdraw(100.0);
        deposit(1000.0);
        System.out.println("Current Balance = "+bal);
        o.withdraw(100.0);
        System.out.println("Current Balance = "+bal);
    }

}
class CheckingAccount{
    void withdraw(double x){
        try{
            BankDemo o = new BankDemo();
            if(o.bal<x)
            throw new InsufficientFundsException("Balance Insufficient");
            else
            o.bal-=x;
        }
        catch(InsufficientFundsException ex){

            System.out.println("error caught = "+ex.getMessage());
        }
    }

}
```

Output:

```
C:\Users\kulvir\Desktop\JAVA CODes\files>javac BankDemo.java

C:\Users\kulvir\Desktop\JAVA CODes\files>java BankDemo
Current Balance = 0.0
error caught = Balance Insufficient
Current Balance = 1000.0
Current Balance = 900.0
```

Question -4

Write a java program to create a Thread by implementing Runnable interface and another thread by extending Thread class. This program will have multiple threads.

Aim:

To create a thread by implementing Runnable interface and another thread by extending thread class.

Algorithm:

1. Start
2. Create a runnable thread that implements runnable
3. Create a run method to start the thread
4. Create an extended thread class
5. Create a run method to start the thread
6. Stop

Code:

```
class RunnableThread implements Runnable {
   Thread thread;
   RunnableThread() {
      thread = new Thread(this, "Thread Runnable");
      System.out.println("The Child Thread: " + thread);
      thread.start();
   }
   public void run() {
```

```
        try {
            for(int i = 0; i < 5; i++) {
                System.out.println("Runnable Child Thread: " + i*10);
                Thread.sleep(1000);
            }
        }
        catch(Exception e) {
            System.out.println("Exception: " + e);
        }
    }
}

class ExtendedThread extends Thread {
    ExtendedThread() {
        super("Thread Extended");
        System.out.println("Child Thread: " + this);
        start();
    }
    public void run() {
        try {
            for(int i = 0; i < 5; i++) {
                System.out.println("Extended Child Thread: " + i);
                Thread.sleep(1000);
            }
        }
        catch(Exception e) {
            System.out.println("Exception: " + e);
        }
    }
```

```
}



public class implement_thread {
    public static void main(String[] args) {
        var runnableThread = new RunnableThread();
        var extendedThread = new ExtendedThread();
        try {
            Thread.sleep(1000);
        }
        catch(Exception e) {
            System.out.println("Exception: " + e);
        }
        System.out.println("Main Thread Completed");
    }
}
```

Output:

```
C:\Users\kulvir\Desktop\JAVA CODes\files>javac implement_thread.java

C:\Users\kulvir\Desktop\JAVA CODes\files>java implement_thread
The Child Thread: Thread[Thread Runnable,5,main]
Child Thread: Thread[Thread Extended,5,main]
Runnable Child Thread: 0
Extended Child Thread: 0
Main Thread Completed
Runnable Child Thread: 10
Extended Child Thread: 1
Extended Child Thread: 2
Runnable Child Thread: 20
Extended Child Thread: 3
Runnable Child Thread: 30
Runnable Child Thread: 40
Extended Child Thread: 4
```