1. File Handling :

```
import java.io.BufferedReader;
import java.io.EOFException;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;

class Person implements Serializable{
	private int no;
	private String name;
	private int sal;
	public Person(int i, String string, int j) {
		// TODO Auto-generated constructor stub
		no=i;
		name=string;
		sal=j;
	}
	public String getName() {
		return name;
	}
	public String toString()
	{
		return no +"  "+ name +"  "+ sal;

	}
}
class CommonFileOperation{
	public void showFiles() {
	  String dirpath = "D:/DAC47/Arun19/Java/Day1/Session2";

		File f3 = new File(dirpath);

		String[] file = f3.list();
		System.out.println("==========================================");
		System.out.println("List of all file in "+dirpath);
		System.out.println("==========================================");
		for (int i = 0; i < file.length; i++) {
			System.out.println(file[i]);
		}
		System.out.println("==========================================");

	}

	public void writeInFile() throws IOException {
		System.out.println("Please write in File");
		FileWriter fw = new FileWriter("dac.txt");

		InputStreamReader in = new InputStreamReader(System.in);
		BufferedReader br = new BufferedReader(in);
```

```
            String str;
            while(!(str = br.readLine()).equals("exit"))
            {
                  fw.write(str+"\n");
            }

            fw.close();
            System.out.println("file write is over");
      }
      public void ReadFromFile() throws IOException {
            FileReader f1 = null;
            try {
                  f1 = new FileReader("dac.txt");
            } catch (FileNotFoundException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            }
            BufferedReader br1 = new BufferedReader(f1);

          String str1;
            while((str1 = br1.readLine())!=null)
          {
            System.out.println(str1);
          }

      }

      public void WriteObjectInFile() throws IOException {
             Person P1=new Person(1,"arun",10000);
             Person P2=new Person(2,"manish",1000);
             Person P3=new Person(3,"Anand",100000);

     FileOutputStream f1 = new FileOutputStream("dac2.txt");
       ObjectOutputStream oo = new ObjectOutputStream(f1);

       oo.writeObject(P1);
       oo.writeObject(P2);
       oo.writeObject(P3);
       System.out.println("object write is over");

      oo.close();
      f1.close();
      }

      public void ReadObjectFromFile() throws IOException,
ClassNotFoundException {

            FileInputStream f2 = new FileInputStream("dac2.txt");
            ObjectInputStream oi = new ObjectInputStream(f2);
            Person obj;

                  try
                  {

            while((obj = (Person)oi.readObject())!=null)
            {
                  System.out.println(obj);
            }
                  }catch(EOFException e)
                  {
                        System.out.println("reached eof");
```

```
                  }

      }
}
public class EXAMFileHandling {
      public static void main(String[] args) throws IOException,
ClassNotFoundException {
            CommonFileOperation qf=new CommonFileOperation();
            qf.writeInFile();
            qf.ReadFromFile();
            qf.WriteObjectInFile();
            qf.ReadObjectFromFile();

      }
}
```

2. JDBC

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;

class JDBC_Statement{
      private Statement smt;
      private Connection con;
      public void MakeConnection() throws ClassNotFoundException, SQLException {
    Class.forName("com.mysql.jdbc.Driver");
    con=DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb?
characterEncoding=latin1","root","arundb");
      System.out.println("connected to data base for simple statement");
            this.smt = con.createStatement();
      }
      public void select() throws SQLException  {
            String q1 = "select * from EMPLOYEES";
            ResultSet rs = smt.executeQuery(q1);

            while(rs.next())
            {
                  System.out.println(rs.getInt(1) +"  "+ rs.getString(2) +"  "+
rs.getString(3));
            }
      }
      public void update() throws SQLException {
                  String q2 = "update EMPLOYEES set  FIRST_NAME = 'aa' where
EMPLOYEE_ID = 3";
            //String q2 = "delete from EMPLOYEES where EMPLOYEE_ID = 3";

             int no=   smt.executeUpdate(q2);
              System.out.println("updatated rows "+no);


      }
```

```
	public void closeDb() throws SQLException {
		con.close();
	}
}




class JDBC_PreparedStatement{
	 private Connection con;
	public void MakeConnection() throws ClassNotFoundException, SQLException {
		Class.forName("com.mysql.jdbc.Driver");
	       con=DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb?
characterEncoding=latin1","root","arundb");
		System.out.println("connected to data base for PREPARED statement");

	}
	public void select() throws SQLException {
		String q1 = "select * from EMPLOYEES where EMPLOYEE_ID = ? ";
		PreparedStatement psmt = con.prepareStatement(q1);
		System.out.println("Please Enter the EMPLOYEE ID...");
		Scanner sc1 = new Scanner(System.in);
		int eno = sc1.nextInt();
		    psmt.setInt(1, eno);
		    ResultSet rs =psmt.executeQuery();

		while(rs.next())
		{
			System.out.println(rs.getInt(1) +"  "+ rs.getString(2) +"  "+
rs.getString(3));
		}

	}
	public void update() throws SQLException {

		String q2 = "update EMPLOYEES set  FIRST_NAME = ? where EMPLOYEE_ID
= ?";
		PreparedStatement psmt = con.prepareStatement(q2);
		Scanner sc1 = new Scanner(System.in);
		System.out.println("Please enter EMPLOYEE First name to
updated...");
		String ename = sc1.nextLine();
		System.out.println("Please Enter the EMPLOYEE ID...");

		int eno = sc1.nextInt();
		      psmt.setString(1, ename);
		    psmt.setInt(2, eno);
		  int no=psmt.executeUpdate();
		  System.out.println(no+" Row updatated Now ");

	}
	public void closeDb() throws SQLException {
		con.close();
	}
}


public class ExamJDBC {
	public static void main(String[] args) throws ClassNotFoundException,
SQLException {
		JDBC_Statement a=new JDBC_Statement();
```

```
		a.MakeConnection();

	System.out.println("==================================================");
		a.select();

	System.out.println("==================================================");
		a.update();

	System.out.println("==================================================");
		a.closeDb();

		JDBC_PreparedStatement a1=new JDBC_PreparedStatement();
		a1.MakeConnection();

	System.out.println("==================================================");
		a1.select();

	System.out.println("==================================================");
		a1.update();

	System.out.println("==================================================");
		a1.closeDb();
	}
}
/*
create table EMPLOYEES(EMPLOYEE_ID int(11),FIRST_NAME varchar(20),LAST_NAME
varchar(20) );
insert into EMPLOYEES values(1,'arun','kumar');
insert into EMPLOYEES values(2,'n','k');
insert into EMPLOYEES values(3,'l','m');*/
```

3. Thread

(1) First

```
class Compute7 extends Thread{
	int x;
	int y;
	int z;
	public void setX(int x) {
		this.x = x;
	}
	public void setY(int y) {
		this.y = y;
	}
	public void setZ(int z) {
		this.z = z;
	}


	public void result() {
```

```
            System.out.println("sin (x) + cos (y) + tan (z) = "+
(Math.cos(this.x)+Math.sin(this.y)+Math.tan(this.z))+" Output By
"+Thread.currentThread().getName());
      }

}
public class ExamThread {

      public static void main(String[] args) throws InterruptedException {
            final Compute7 a = new Compute7();

            Thread t1=new Thread(a)
            {
                  public void run()
                  {
                        a.setX(10);
                  }
            };


            Thread t2 =new Thread(a)
            {
                  public void run()
                  {
                        a.setY(20);
                  }
            };


            Thread t3=new Thread(a)
            {
                  public void run()
                  {
                        a.setZ(30);
                  }
            };
            t1.start();
            t2.start();
            t3.start();
            t1.join();
            t2.join();
            t3.join();
            Thread t4=new Thread(a)
            {
                  public void run()
                  {
                        a.result();
                  }
            };
            t4.start();
      }

      }
```

(2) Second

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
```

```
class R7 extends Thread{
	String fname;
	//FileReader f1;  //"threadTxt1.txt"
	BufferedReader br1;
	static boolean IsSleep1=true;


	public void setFname(String fname) {
		this.fname = fname;
	}


	public synchronized void ReadFirstFile(String fname) throws IOException,
InterruptedException {
		String str1;
		FileReader f1=null;
		BufferedReader br1=null;
		try {
			f1 = new FileReader(fname);
			br1 = new BufferedReader(f1);
			while((str1 = br1.readLine())!=null)
				{
					System.out.println(str1+"    -> By Thread
"+Thread.currentThread().getName());
					if (IsSleep1) {
						IsSleep1=false;
						notify();
						wait();
					}
				}
		} catch (FileNotFoundException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
		finally{
			br1.close();
		    f1.close();
		}


	}
     public synchronized void ReadSecondFile(String fname) throws IOException,
InterruptedException {
	   String str1;
		FileReader f1=null;
		BufferedReader br1=null;
		try {
			f1 = new FileReader(fname);
			br1 = new BufferedReader(f1);
			while((str1 = br1.readLine())!=null)
				{
					System.out.println(str1+"    -> By Thread
"+Thread.currentThread().getName());
					if (!IsSleep1) {
						IsSleep1=true;
						notify();
						wait();
```

```
                        }
                    }
            } catch (FileNotFoundException e) {

                  e.printStackTrace();
            } catch (IOException e) {

                  e.printStackTrace();
            }
            finally{
                  br1.close();
                f1.close();
            }
      }


}
public class ExamThread1 {

      public static void main(String[] args) throws InterruptedException {

     final R7 t1 = new R7();

     Thread s1=new Thread(t1)
            {
                  public void run()
                  {
                        try {
                              t1.ReadFirstFile("threadTxt1.txt");
                        } catch (IOException e) {
                              // TODO Auto-generated catch block
                              e.printStackTrace();
                        } catch (InterruptedException e) {
                              // TODO Auto-generated catch block
                              e.printStackTrace();
                        }
                  }
            };


            Thread s2=new Thread(t1)
            {
                  public void run()
                  {
                        try {
                              t1.ReadSecondFile("threadTxt2.txt");
                        } catch (IOException e) {
                              // TODO Auto-generated catch block
                              e.printStackTrace();
                        } catch (InterruptedException e) {
                              // TODO Auto-generated catch block
                              e.printStackTrace();
                        }
                  }
            };
            s1.start();
            s2.start();
            s1.join();
            s2.join();
            s1.stop();
          s2.stop();
```

```
      }

}
```

(3) Third

```
class Table{
      synchronized void printTable(int n){//method not synchronized
   for(int i=1;i<=5;i++){
     System.out.println(n*i);
     try{
      Thread.sleep(400);
     }catch(Exception e){System.out.println(e);}
   }

 }
}

class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}

}
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}

public class ExamThread3{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

4 . Collection ArrayList

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
```

```java
import java.util.Iterator;


public class ExamCollection {

	public static void main(String[] args) {

	ArrayList<Integer> al=new ArrayList<>();


		// adding elements
		System.out.println("adding elements ");
		for(int i=0;i<10;i++){
			al.add(i+100);
		}

		System.out.println(al);

		// remove element by index
	System.out.println("emove element by index index 3");
		al.remove(3);
		System.out.println(al);

		// remove element by value
		System.out.println("remove element by value  108");

				al.remove(al.indexOf(108));
				System.out.println(al);


		System.out.println("Modify value 109 by 1009");
		int in=al.indexOf(109);
		al.remove(in);
		al.add(in,1009);
		System.out.println(al);


		// get particular element
				System.out.println(" get particular element by index
3");
		System.out.println(al.get(3));

		System.out.println("Print using Iterator");
		Iterator it = al.iterator();
		while(it.hasNext())
	    {
		System.out.println(it.next());
	    }

		System.out.println("print using for loop");
		for(int i=0;i<al.size();i++){
			System.out.println(al.get(i));
		}
		System.out.println("print after sorting");
		Collections.sort(al);
		System.out.println(al);



		// To clear all elements
		al.clear();
		System.out.println(al);
```

```
    }
}
```

5. Method Of ArrayList

forEach(Consumer action): Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception.

1. retainAll(Collection c): Retains only the elements in this list that are contained in the specified collection.
2. removeIf(Predicate filter): Removes all of the elements of this collection that satisfy the given predicate.
3. contains(Object o): Returns true if this list contains the specified element.
4. remove(int index): Removes the element at the specified position in this list.
5. remove(Object o): Removes the first occurrence of the specified element from this list, if it is present.
6. get(int index): Returns the element at the specified position in this list.
7. subList(int fromIndex, int toIndex): Returns a view of the portion of this list between the specified fromIndex, inclusive, and toIndex, exclusive.
8. spliterator(): Creates a late-binding and fail-fast Spliterator over the elements in this list.
9. set(int index, E element): Replaces the element at the specified position in this list with the specified element.
10. size(): Returns the number of elements in this list.
11. removeAll(Collection c): Removes from this list all of its elements that are contained in the specified collection.
12. ensureCapacity(int minCapacity): Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.

13. listIterator(): Returns a list iterator over the elements in this list (in proper sequence).
14. listIterator(int index): Returns a list iterator over the elements in this list (in proper sequence), starting at the specified position in the list.
15. isEmpty(): Returns true if this list contains no elements.
16. removeRange(int fromIndex, int toIndex): Removes from this list all of the elements whose index is between fromIndex, inclusive, and toIndex, exclusive.
17. void clear(): This method is used to remove all the elements from any list.
18. void add(int index, Object element): This method is used to insert a specific element at a specific position index in a list.
19. void trimToSize(): This method is used to trim the capacity of the instance of the ArrayLis to the list's current size.
20. int indexOf(Object O): The
index the first occurrence of a specific element is either returned, or -1 in case the element is not in the list.
21. int lastIndexOf(Object O): The index the last occurrence of a specific element is either returned, or -1 in case the element is not in the list.
22. Object clone(): This method is used to return a shallow copy of an ArrayList.
23. Object[] toArray(): This method is used to return an array containing all of the elements in the list in correct order.
24. Object[] toArray(Object[] O): It is also used to return an array containing all of the elements in this list in the correct order same as the previous method.
25. boolean addAll(Collection C): This method is used to append all the elements from a specific collection to the end of the mentioned list, in such a order that the values are returned by the specified collection's iterator.
26. boolean add(Object o): This method is used to append a specificd element to the end of a list.

27.boolean addAll(int index, Collection C): Used to insert all of the elements starting at the specified position from a specific collection into the mentioned list.

6. hashmap Methods

void clear(): Used to remove all mappings from a map.

- boolean containsKey(Object key): Used to return True if for a specified key, mapping is present in the map.
- boolean containsValue(Object value): Used to return true if one or more key is mapped to a specified value.
- Object clone(): It is used to return a shallow copy of the mentioned hash map.
- boolean isEmpty(): Used to check whether the map is empty or not. Returns true if the map is empty.
- Set entrySet(): It is used to return a set view of the hash map.
- Object get(Object key): It is used to retrieve or fetch the value mapped by a particular key.
- Set keySet(): It is used to return a set view of the keys.
- int size(): It is used to return the size of a map.
- Object put(Object key, Object value): It is used to insert a particular mapping of key-value pair into a map.
- putAll(Map M): It is used to copy all of the elements from one map into another.
- Object remove(Object key): It is used to remove the values for any particular key in the Map.
- Collection values(): It is used to return a Collection view of the values in the HashMap.