```java
public class EmployeeDetails {

    public List<String> getEmployees() {

        List<String> emps = new ArrayList<>();
        emps.add(e: "1-ABC-SDE1-9923291373");
        emps.add(e: "2-DEF-SDE2-3729475821");
        emps.add(e: "3-HIJ-SDE3-1539464506");
        emps.add(e: "4-KLM-SDE4-9927364009");
        emps.add(e: "5-NOP-SDE5-7226394001");
        emps.add(e: "6-QRS-SDE6-2771663721");
        emps.add(e: "7-TUV-SDE7-8301727421");
        emps.add(e: "8-WXY-SDE8-2131554367");

        return emps;
    }
}
```

Source →

```java
public class EmployeeToPLAdapter implements PhoneListSource {
    EmployeeDetails empdetails;

    public EmployeeToPLAdapter(EmployeeDetails empdetails) {
        this.empdetails = empdetails;
    }

    @Override
    public List<String> getPhoneNumbers() {
        // TODO Auto-generated method stub
        List<String> details = empdetails.getEmployees();
        List<String> phoneNos = new ArrayList<>();

        for(String detail:details){
            String[] parts = detail.split(regex: "-");
            phoneNos.add(parts[3]);
        }

        return phoneNos;
    }
}
```

Adapter →

```java
public class Intranet {
    PhoneListSource source;

    Intranet(PhoneListSource source){   (adapter)
        this.source = source;
    }

    public void printPhoneNumbers(){
        List<String> numbers = source.getPhoneNumbers();
        System.out.println(numbers);
    }
}
```

Output

```java
public interface PhoneListSource {
    public List<String> getPhoneNumbers();
}
```

```java
public class Test {
    Run | Debug
    public static void main(String[] args) {
        EmployeeDetails dataSource = new EmployeeDetails();
        EmployeeToPLAdapter adapter = new EmployeeToPLAdapter(dataSource); //Adapter needs datasource for conversion
        Intranet intra = new Intranet(adapter); //we can pass Class object which implements
        intra.printPhoneNumbers();              // Interface PhoneListSource to Intranet.
    }
}
```

```java
public class Test {
    Run | Debug
    public static void main(String[] args) {
        EmployeeDetails dataSource = new EmployeeDetails();
        EmployeeToPLAdapter adapter = new EmployeeToPLAdapter(dataSource); //Adapter needs datasource for conversion
        Intranet intra = new Intranet(adapter); //we can pass Class object which implements
        intra.printPhoneNumbers();              // Interface PhoneListSource to Intranet.
    }
}
```

```java
import java.util.List;

public interface PhoneListSource {
    public List<String> getPhoneNumbers();
}
```

Employee details → data source (raw data)

EmployeeToPL Adapter → gets source for conversion

EmployeeToPL Adapter → Implements phoneListSource interface

" " → Overrides getPhoneNumbers & returns converted data.

Intranet → Accepts EmployeeToPLAdapter adapter object.
Calls adapter.getPhoneNumbers internally & prints intra.printPhnNos.

Data Source class :- Employee details.

↓

Injected into Adapter (for conversion)

↓

Adapter implements Interface whose function returns converted data.

(has its own internal logic)

Adapter overrides this function to return this required converted data

↓

Intranet Accepts this source through class's object who implements the interface. (this object is passed through dependency.)

↓

Intranet handles this converted data and prints it out.

Adapter

-> There is an already existing data provider which provides data in format1

-> There is an already existing data consumer which expects data in format2

-> format1 != format2

-> We have to write an adapter which converts between the two formats

-> Consumer will call the adapter

-> Adapter will call the provider and get data in format1

-> Adapter will convert data from format1 to format2

-> Consumer will get data in format2 from the adapter

Usecase:- API's data conversion.

Singleton :- Used for Logs.

only allows 1 object of that class.

```java
public class Singleton {
    private static Singleton inst;

    private Singleton() {
    }

    public static Singleton getTheSingleInstance() {
        if(inst == null) {
            synchronized (Singleton.class) {
                if(inst == null) {
                    inst = new Singleton();
                }
            }
        }

        return inst;
    }
}
```

→ private constructor

```java
public class Test {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        Singleton s1 = Singleton.getTheSingleInstance();
        Singleton s2 = Singleton.getTheSingleInstance();

        System.out.println(s1 == s2);
    }

}
```

static reference.

Private constructor.

Synchronized for handling multithreading case.