```
interface StudentService{


public void registerStudent(Student student);

public Student getStudentByRoll(int roll);

public boolean updateStudent(Student student);

}


class StudentServiceImpl1 implements StudentService{

public void registerStudent(Student student){

//using JDBC approach
}

public Student getStudentByRoll(int roll){
//using JDBC approach
}

public boolean updateStudent(Student student);
//using JDBC approach
}


class StudentServiceImpl2 implements StudentService{

public void registerStudent(Student student){

//using ORM approach
}

public Student getStudentByRoll(int roll){
//using ORM approach
}

public boolean updateStudent(Student student);
//using ORM approach
}
```

```
@Autowire
StudentService service;

service.registerStudent(student);
service.getStudentByRoll(10);
service.updateStudent(student);
```

example 2:

```
interface X {

void funX();

}

interface Y {

void funY();

}
```

Note: one class can extends another class and implements multiple interfaces simulteniously.
example

```
class A extends B implements X,Y{


}
```

Note: we can have an empty interface also.

```
public interface X{

}
```

--this type of empty interface is also known as tag or marker interface.

--some of the predefined marker interfaces in Java are:

java.io.Serializable
java.lang.Cloanable

--these kind of marker interfaces used to specify certain type of special behaviour of our object.

```
class Student implements Serializable{

}
```

Student s1=new Student(10,"Ram",780); // moving out this Student obj from the RAM to outside the RAM is known as serialization. and reverse is known as deserialization.

--if we want to group multiple objects to a special type then also we can make use Marker interface.

example:

Special.java:-

```
package com.masai;

//marker interface
public interface Special {

}
```

A.java:

```
package com.masai;

public  class A implements Special{
     void funA() {

               System.out.println("inside funA of A");
          }


}
```

Demo.java:

```
package com.masai;

public class Demo {


          public void fun1(Special special) {

                    //want to specify some specail behaviour

                    System.out.println("here special behaviour is applied");

          }


          public static void main(String[] args) {

                    Demo d1= new Demo();

                    d1.fun1(new A());

          }
}
```

Collection framework:
==================

Collection: if we want to represent a group of objects as a single unit (single object) then we should use collection.

--Collection of objects..

Framework:

--the main objective of a f/w is ease developer work.

--it is semimplemented architecture.

--A f/w comprises some abstract design with built-in behaviour(functionality), in order to use it, we need to insert our functionality in various places of f/w.

--A s/w f/w is a universal, resuable s/w platform to develop s/w applications, products or solutions.

Data structure:

--array
--stack
--queue
--LinkedList
--searching
--sorting

Collection f/w:
============

--it defines several classes and interfaces which can be used to represent/arrange group of objects as single unit(object)

--each collection classes are like a container or a bag, where we can hold multiple objects.

--it is basically readymade implemented DS.

---we can group/hold/store multiple object as a single object by using array.

--in Java array is also one type of object

--5 student object as a single object.

Student[] students = new Student[5]; // here one array object is created and 6 variables are created.

students // ref type array variable

student[0] // ref type Student class variable
student[1] // ref type Student class variable
student[2] // ref type Student class variable
student[3] // ref type Student class variable
student[4] // ref type Student class variable

student[0] = new Student(10,"N1",780);
student[1] = new Student(12,"N2",880);
student[2] = new Student(13,"N3",680);
student[3] = new Student(14,"N4",580);
student[4] = new Student(15,"N5",680);

limitations of array:
----

1. size is fixed, we can not increase or decrease it dynamically.

2.it supports homogenious type of Data/element.

--this limitation we can overcome by taking Object class array.

Object[] objs= new Object[3];

objs[0]=new Student();
objs[1]=new Employee();
objs[2]=new Product();

3. array concept is not implemented based on readymade method support.
for each and every requirement we need to write our own logic,.
even for printing the elements from an array we need to write for loop.

--in each array object we have only one non-static variable i.e 'length'.

--to overcome the above limitation we need to use collection f/w.

feature of Collection f/w:
====================

--collections are growable and shinkable in nature.

--collection can hold both homogenoius and heterogenious type of element.

--every collection classes are implemented based on some standard data-structure, hence readymade method
support is available for most of the requirements.

*****All(99%) the collection f/w related classes and interfaces belongs to java.util package.**

--collection f/w having 2 section:

1. normal collection (here we manage object uniformally/individually/ singular manner).

2. Map (here we manage objects in key-value pair)

List: ---> when we need to arrange the elements in a sequence (index based manner) and duplicate elements are allowed.

Set: -- when we need uniqueness (duplicate elements are not allowed)

Queue:-- when we need to arrange the elements prior to processing(FIFO order is default, but we can manipulate)

******Note:- from Java 1.5 LinkedList class also implements Deque interface.**

Collection Interface:
================

--It is the foundation upon which the collection f/w is built.

--it declares some of the core methods that all collection classes will have.

--these methods we can call on any collection f/w related classes objects.

some of methods of collection interface:
==================================

int size(); // how many elements are there in that collection object.

boolean isEmpty();

boolean contains(Object obj); //for searching any element.

Iterator iterator(); // it is inherited from Iterable interface

Object[] toArray();//to convert any collection object to normal array

T[] toArray(T[]); //convert only limited size element to array

boolean add(Object obj); // it is a most commonly used method, to add any element insdie any tupe of
collection.
boolean remove(Object obj);

boolean containsAll(Collection col);

boolean addAll(Collection col); copy all the element from one collection to our collection object
boolean removeAll(Collection col);

boolean retainAll(Collection col); // except supplied collection elements all other will be removed.
void clear(); // remove all the elements from the collectioj

// these following 2 methods belongs to Object class
boolean equals(Object obj); //compare two collection object
int hashCode(); // return the hashCode of collection obj

//these methods introduced in java 1.8 to perform functional programming.
public boolean removeIf(java.util.function.Predicate<? super E>);
public java.util.Spliterator<E> spliterator();
public java.util.stream.Stream<E> stream();
public java.util.stream.Stream<E> parallelStream();