SOLID principles assignment

S – Single Responsibility principle

```
public class BookingService {

public static ServiceProvider book(ArrayList<ServiceProvider> searchResult) {

Scanner scanner = new Scanner(System.in);
ServiceProvider chosenServiceProvider=null;

System.out.println("Choose an option : ");
int optionChosen = scanner.nextInt();

chosenServiceProvider=searchResult.get(optionChosen-1);

return chosenServiceProvider;
}
}
```

O – open closed principle

```
public class Transportation implements ProviderEditable,AdminEditable {
private String transportationType;
private String serviceID;
private String serviceName;
public String getTransportationType() {
return transportationType;
}
public void setTransportationType(String transportationType) {
this.transportationType = transportationType;
}
public String getServiceID() {
return serviceID;
}
public void setServiceID(String serviceID) {
this.serviceID = serviceID;
}
public String getServiceName() {
return serviceName;
}
public void setServiceName(String serviceName) {
this.serviceName = serviceName;
}
public Transportation(String transportationType, String serviceID, String serviceName) {
```

```
super();
this.transportationType = transportationType;
this.serviceID = serviceID;
this.serviceName = serviceName;
}
}


public class Cruise extends Transportation {
private String cruiseNumber;

public String getCruiseNumber() {
return cruiseNumber;
}

public void setCruiseNumber(String cruiseNumber) {
this.cruiseNumber = cruiseNumber;
}

public Cruise(String transportationType, String serviceID, String serviceName, String
cruiseNumber) {
super(transportationType, serviceID, serviceName);
this.cruiseNumber = cruiseNumber;
}
}




public class Train extends Transportation {
private String trainNumber;

public String getTrainNumber() {
return trainNumber;
}

public void setTrainNumber(String trainNumber) {
this.trainNumber = trainNumber;
}

public Train(String transportationType, String serviceID, String serviceName, String
trainNumber) {
super(transportationType, serviceID, serviceName);
this.trainNumber = trainNumber;
}
}
public class SearchService {

public static ServiceProvider search(ArrayList<ServiceProvider> providerList) {
Scanner scanner = new Scanner(System.in);

System.out.println("------------ Search------------");
```

```
System.out.println("Enter string to search : ");

String searchString = scanner.next();

ServiceProvider chosenServiceProvider=null;

ArrayList<ServiceProvider> searchResult = new ArrayList<ServiceProvider>();

int option=1;
for ( ServiceProvider serviceProvider : providerList) {
if (serviceProvider.getTransportation().getServiceName().contains(searchString)||
serviceProvider.getTransportation().getTransportationType().contains(searchString)) {
searchResult.add(serviceProvider);
System.out.println(option+" : ");
DisplayServiceProvider.display(serviceProvider);
option++;
}
}

if (searchResult.size()==0) {
return null;
}

chosenServiceProvider=BookingService.book(searchResult);

return chosenServiceProvider;
}
}
```

Now, Search function will continue to work even if we add another class which extends transportation as follows

```
public class Flight extends Transportation {
private String flightNumber;

public String getFlightNumber() {
return flightNumber;
}

public void setFlightNumber(String flightNumber) {
this.flightNumber = flightNumber;
}

public Flight(String transportationType, String serviceID, String serviceName, String
flightNumber) {
super(transportationType, serviceID, serviceName);
this.flightNumber = flightNumber;
}
}
```

L - Liskov Substitution Principle

```
transportation= (Transportation) serviceProvider.getTransportation();
```

Transportation can contain the objects of any class that extends Transportation class like Cruise, Taxi etc.

I - Interface Segregation Principle

```
public interface ProviderEditable {
public String getServiceName();
public void setServiceName(String serviceName);
public String getServiceID();
public String getTransportationType();
}

public interface AdminEditable {
public String getTransportationType();
public String getServiceID();
public void setServiceID(String serviceID);
public String getServiceName();
}
```

Here, ProviderEditable and AdminEditable are two interfaces. They could have been combined together, but this helps the logic.-

D - Dependency Inversion Principle

```
public class Transportation implements ProviderEditable,AdminEditable {
private String transportationType;
private String serviceID;
private String serviceName;
public String getTransportationType() {
return transportationType;
}
public void setTransportationType(String transportationType) {
this.transportationType = transportationType;
}
public String getServiceID() {
return serviceID;
}
public void setServiceID(String serviceID) {
this.serviceID = serviceID;
```

```
}
public String getServiceName() {
return serviceName;
}
public void setServiceName(String serviceName) {
this.serviceName = serviceName;
}
public Transportation(String transportationType, String serviceID, String serviceName) {
super();
this.transportationType = transportationType;
this.serviceID = serviceID;
this.serviceName = serviceName;
}
}


public class ServiceProvider extends User {
private String serviceProviderID;
private ProviderEditable transportation;
public String getServiceProviderID() {
return serviceProviderID;
}
public void setServiceProviderID(String serviceProviderID) {
this.serviceProviderID = serviceProviderID;
}
public ProviderEditable getTransportation() {
return transportation;
}
public void setTransportation(ProviderEditable transportation) {
this.transportation = transportation;
}
public ServiceProvider(String username, String serviceProviderID, ProviderEditable
transportation) {
super(username);
this.serviceProviderID = serviceProviderID;
this.transportation = transportation;
}
}
```

And in the utility package, inside the main function,

```
...
System.out.println("Enter Id of Service provider : \n");
String serviceProviderIDtoEdit = scanner.next();
AdminEditable transportation =null;

for( ServiceProvider serviceProvider : providerList) {

       if(serviceProvider.getServiceProviderID().compareTo(serviceProviderIDtoEdit)==0)
{
transportation= (Transportation) serviceProvider.getTransportation();
break;
          }
```

```
}
...
```

Here, we have removed the dependancy between `ServiceProvider` and Admin user. We can also give access to specific functions using dependency inversion principle .