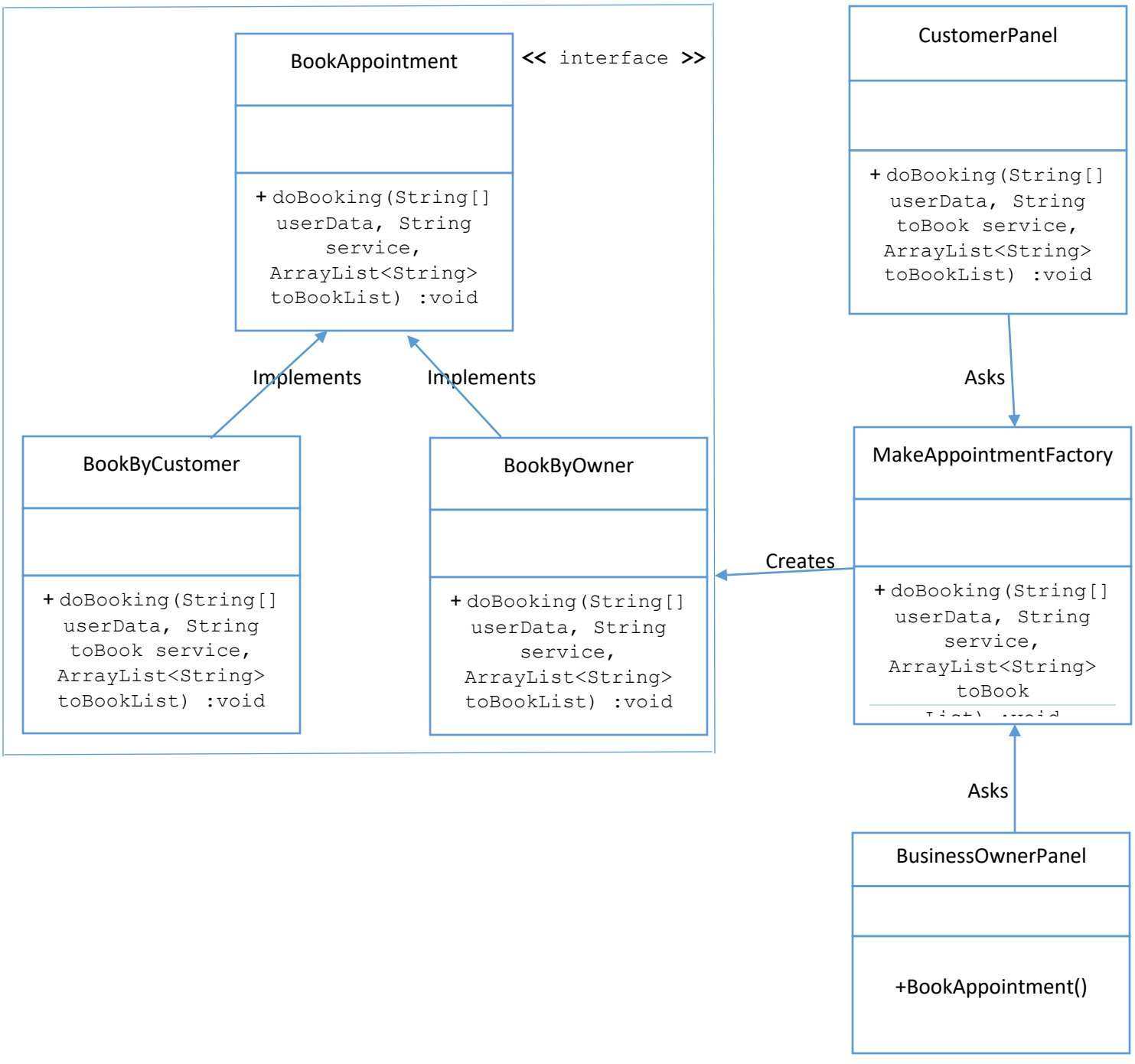
Factory Design Pattern

In factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface. We define an interface or abstract class for creating an object but let the subclasses decide which class to instantiate. In other words subclasses are responsible to create the instance of the class.

We have created a **`BookAppointment`** interface and concrete classes implementing the **`BookAppointment`** interface.

We have defined a factory class called **`MakeAppointmentFactory`**



Book Appointment Factory Method:

Step 1. Create a **`BookAppointment`** `interface`

```
package utility;
import java.util.ArrayList;
public interface BookAppointment {
        public void doBooking(String[] userData, String service, ArrayList<String> toBookList);
}
```

Step 2. Create the concreate classes that implements the **`BookAppointment`** `interface`

```
package utility;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.util.ArrayList;
import javax.swing.JOptionPane;

public class BookByCustomer implements BookAppointment {

public BookByCustomer() {
}

  public void doBooking(String[] userData, String service, ArrayList<String> toBookList) {
ArrayList<String> list = new ArrayList<>();
String fileName = service + ".txt";
String selected = "";
for (int i = 0; i < toBookList.size(); i++) {
try {
selected = toBookList.get(i);
```

```
String recs[];
while (true) {
recs = selected.split(",");
if (recs[4].equals("Un-available")) {
System.out.println("Booking Un-Available");
System.out.println("Please select another");
} else {
recs[4] = "Un-available";
break;
}
}
String modified = recs[0] + "," + recs[1] + "," + recs[2] + ","+ recs[3] + "," + recs[4];
BufferedReader br = new BufferedReader(new FileReader(fileName));
String line = "";
while ((line = br.readLine()) != null) {
list.add(line);
}
BufferedWriter bw = new BufferedWriter(new FileWriter(fileName));
for (int a = 0; a < list.size(); a++) {
if (list.get(a).equals(selected)) {
list.set(a, modified);
 bw.write(list.get(a));
 bw.newLine();
} else {
bw.write(list.get(a));
 bw.newLine();}
}
bw.close();
BufferedWriter writer2 = new BufferedWriter(new FileWriter("BookingSummaries.txt", true));
```

```
writer2.write("Customer," + userData[0] + "," + userData[1] + ",booked Appointment on," + recs[0] + "," + recs[1] + "," + /* servicename */service + "," + recs[2] + ","+ recs[3]);

 writer2.newLine();

writer2.close();

} catch (Exception e) {

e.printStackTrace();

}

}

JOptionPane.showMessageDialog(null, "Successfully Booked By Customer");

}

}
```

//BookByOwner Code Example Goes Here…….

Step 3. Create a MakeAppointmentFactory to generate object of concreate classes based on given data.

It generates object for BookByOwner if given user type as Owner and generates object for BookByCustomer if given user type as Customer.

```
package utility;

import java.util.ArrayList;

import javax.swing.JOptionPane;

public class MakeAppointmentFactory {

public MakeAppointmentFactory() {

}

public static void doBooking(String userType, String[] userData, String service, ArrayList<String> toBookList){

 if(userType.equals("Customer")){

new BookByCustomer().doBooking(userData, service, toBookList);
```

```
}else if(userType.equals("Owner")){
new BookByOwner().doBooking(userData, service, toBookList);
}
}
}
```