

**Return-Campus Resident Management System**

Course Code: CPS 2232\_W05\_Fall 2021

Course Name: Data Structures

Student Group List:

Student Name: Wang Junfeng (Claire) Student Number: 1129618

Student Name: Liu Meili Student Number: 1192738

Student Name: Wu Kangmin Student Number: 1162949

Professor: Dr Ken Ehimwenma, Ph.D.

Date: December 13, 2021

**Abstract**

This project is a simple simulation of WKU’s return-campus management system. As a respond to the campus epidemic prevention and control requirements, this system enables administrators to easily process return-campus applications, arrange dormitory distribution, and manage information database which can be completed by simple button operations.­

­­­­­­­­­­­­­­­­­

1. **Introduction**

The return-campus management system consists of two main functions:

(1) Record return-campus applications into database. Two separated portals are designed for students and faculties respectively.

(2) Modifying resident and room information, including record of arrival, changing rooms, put high-risk students into quarantine, and reject return-campus applications.

Java Swing & AWT are used to develop a graphical interface to display room and resident information. Data can be managed easily through different function buttons.

Apache Derby is chosen to be our database tool. Derby database is accessed into our project via an embedded JDBC driver.

Function (1) is to record the information of each resident and update the status of the room (available or not). After filling in the information, the system will query the database to search certain rooms according to the type of room they want to stay (dormitory types such as single bed at Plum Hall, 2-Beds at Orchid Hall). If the room status is available, the administrator will arrange the applicant to the room according to the applicant’s gender. If there is no room, the check-in fails.

Function (2) is designed for administrator to manage the two databases collaboratively: database 1(residents) and database 2(room). Under function (2), there are 3 sub-functions: mark as arrive, transfer room, and cancel.

* Mark as arrive means allow one’s room choice, the system will simply change the room status as “Checked In”.
* Transfer will enable administrator to search and retrieve a person’s information by one’s unique ID number. The room transfer operation will first check a certain room’s availability, and the transfer will be successful if the room status matches the “available” keyword. After the room transfer, the original room will be marked as “available”. The new room will be marked as “Checked In” at the same time.
* Cancel means reject one’s application if there is no available room of the required type or the student come from high-risk epidemic areas. This operation needs to be confirmed again.

1. **Methodology**
2. Singleton Method: Ensure that after a class is created, a global access is provided to the object.
3. Factory Method: Defines an interface for creating objects but let subclasses to decide which class to instantiate and refers to the newly created object through a common interface.
4. Builder Method: Defines an instance for creating an object but lets subclasses decide which class to instantiate and allows finer control over the construction process.
5. Observer Method: Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

**a). UML Diagrams**

**b). UML Code Generation**

**c). Full Code Implantation (and Screenshots)**

1. package ProjectCode;
3. import java.awt.event.\*;
4. import java.sql.\*;
5. import javax.swing.\*;
7. **public** **class** Main {
8. **public** **static** **void** main(String[] args) {
9. SystemFrame a = **new** SystemFrame() ;
10. }
11. }
12. package ProjectCode;
14. import java.awt.event.\*;
15. import java.sql.\*;
16. import javax.swing.\*;
18. @SuppressWarnings("serial")
19. **public** **class** SystemFrame extends JFrame implements ActionListener {
20. JButton bookingFunction ;           //register (book) function
21. JButton arriveFunction ;            //mark as arrive function
22. JButton changeFunction ;            //change function
23. JButton back ;                      //back button
24. JPanel home ;                       //home page
25. RegisterPanel registerPanel ;       //student register panel
26. RHDPanel rhdPanel ;                 //RHD panel
27. **int** flag = 0 ;                      //mark which panel is used
28. SystemFrame() {
29. setLayout(null) ;
30. setBounds(300, 100, 800, 800);
31. setVisible(**true**) ;
32. setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE) ;
33. bookingFunction = **new** JButton("CheckIn [for Students & Faculty]") ;
34. bookingFunction.addActionListener(**this**) ;
35. bookingFunction.setBounds(100, 300, 250, 100);
36. arriveFunction = **new** JButton("Management [for RHD]") ;
37. arriveFunction.addActionListener(**this**) ;
38. arriveFunction.setBounds(450, 300, 250, 100);
39. back = **new** JButton("Return") ;
40. back.setBounds(100, 700, 100, 50) ;
41. back.addActionListener(**this**) ;
42. back.setVisible(**false**) ;
43. home = **new** JPanel() ;
44. home.setLayout(null) ;
45. home.setBounds(0, 0, 800, 800) ;
46. home.add(bookingFunction) ;
47. home.add(arriveFunction) ;
48. add(back) ;
49. add(home) ;
50. }
51. **public** **void** actionPerformed(ActionEvent e) {
52. **if**(e.getSource() == bookingFunction) {      //click to enter the function
53. registerPanel = **new** RegisterPanel() ;
54. home.setVisible(**false**) ;
55. remove(home) ;
56. add(registerPanel) ;
57. back.setVisible(**true**) ;
58. flag = 1 ;
59. }
60. **if**(e.getSource() == arriveFunction) {       //click to enter the function
61. rhdPanel = **new** RHDPanel() ;
62. home.setVisible(**false**) ;
63. remove(home) ;
64. add(rhdPanel) ;
65. back.setVisible(**true**) ;
66. flag = 2 ;
67. }
68. **if**(e.getSource() == back) {             //back to previous page
69. **if**(flag == 1) {
70. registerPanel.setVisible(**false**) ;
71. remove(registerPanel) ;             //remove text box
72. }
73. **if**(flag == 2) {
74. rhdPanel.setVisible(**false**) ;        //remove RHD panel
75. remove(rhdPanel) ;
76. }
77. home.setVisible(**true**) ;
78. add(home) ;                         //show buttons again
79. back.setVisible(**false**) ;
80. }
81. }
82. }
83. package ProjectCode;
85. **public** **class** Apartment implements Room {
86. String type ;       //room type
87. String state ;      //room state
88. **int** roomNum ;       //æˆ¿å�·
89. **double** price ;      //ä»·æ ¼
90. Apartment() {}
91. Apartment(String type, **int** roomNum, String state, **double** price) {
92. **this**.type = type ;
93. **this**.roomNum = roomNum ;
94. **this**.state = state ;
95. **this**.price = price ;
96. }
97. **public** **void** setState(String state) {    //ä¿®æ”¹æˆ¿é—´çŠ¶æ€�
98. **this**.state = state ;
99. }
100. **public** **double** getPrice() {              //è¿”å›žä»·æ ¼
101. **return** price ;
102. }
103. **public** **int** getRoomNum() {               //è¿”å›žæˆ¿å�·
104. **return** roomNum ;
105. }
106. **public** String getType() {               //è¿”å›žæˆ¿é—´ç±»åž‹
107. **return** type ;
108. }
109. **public** String getState() {              //è¿”å›žæˆ¿é—´çŠ¶æ€�
110. **return** state ;
111. }
112. }
113. package ProjectCode;
114. import java.awt.event.\*;
115. import java.sql.\*;
116. import javax.swing.\*;
118. **public** **class** Cancel {
119. Connection con ;
120. Statement sql ;
121. ResultSet rs ;
122. Cancel(String phone) {
123. del(phone) ;
124. }
125. **void** del(String ID) {       //cancel application
126. **try** {   Class.forName("org.apache.derby.jdbc.EmbeddedDriver") ; }
127. **catch**(Exception e) {}
128. **try** {
129. con = DriverManager.getConnection("jdbc:derby:data;create=false") ;
130. sql = con.createStatement() ;
131. rs = sql.executeQuery("SELECT \* FROM bookingMessage WHERE ID='"+ID+"'") ;   //search by id
132. rs.next() ;                         //Jump to the next one, which is the search result
133. **int** flag = JOptionPane.showConfirmDialog(**new** RHDPanel(), "Do you want to delete itï¼Ÿ", null, JOptionPane.YES\_NO\_CANCEL\_OPTION) ;
134. **if**(flag == JOptionPane.YES\_OPTION) {
135. **int** roomNum = rs.getInt(2) ;        //Get room number
136. sql.executeLargeUpdate("DELETE FROM bookingMessage WHERE ID='"+ID+"'") ;
137. sql.executeLargeUpdate("UPDATE roomInf SET state='available' WHERE roomNum="+roomNum) ; //update room status
138. JOptionPane.showMessageDialog(**new** RHDPanel(), "Delete success", null, JOptionPane.INFORMATION\_MESSAGE);
139. }
140. con.close();
141. }
142. **catch**(Exception e) {
143. System.out.print(e) ;
144. JOptionPane.showMessageDialog(**new** RHDPanel(), "Delete failed", null, JOptionPane.INFORMATION\_MESSAGE);
145. }
146. }
147. }
148. package ProjectCode;
149. import java.sql.\*;
150. import javax.swing.\*;
152. **public** **class** Change {
153. Connection con ;
154. Statement sql ;
155. ResultSet rs ;
156. Change(String ID, **int** newNum) {
157. transfer(ID, newNum) ;
158. }
159. **void** transfer(String ID, **int** newNum) {      //Update check in status
160. **try** {   Class.forName("org.apache.derby.jdbc.EmbeddedDriver") ; }
161. **catch**(Exception e) {}
162. **try** {
163. con = DriverManager.getConnection("jdbc:derby:data;create=false") ;
164. sql = con.createStatement() ;
165. rs = sql.executeQuery("SELECT \* FROM bookingMessage WHERE ID='"+ID+"'") ;//Search by id
166. rs.next() ;             //Jump to the next one(the search result)
167. //String type = rs.getString(6) ;       //Room type
168. **int** befortNum = rs.getInt(2) ;         //Original room number
169. rs = sql.executeQuery("SELECT \* FROM roomInf WHERE roomNum="+newNum) ;
170. rs.next() ;
171. String state = rs.getString(3) ;
172. **if**( state.equals("available") ) {
173. sql.executeLargeUpdate("UPDATE bookingMessage SET roomNum="+newNum+" WHERE roomNum="+befortNum) ;
174. sql.executeLargeUpdate("UPDATE roomInf SET state='available' WHERE roomNum="+befortNum) ;//Update before room status
175. sql.executeLargeUpdate("UPDATE roomInf SET state='reserved' WHERE roomNum="+newNum) ;   //Update new room status
176. JOptionPane.showMessageDialog(**new** RHDPanel(), "Transfor successfully", null, JOptionPane.INFORMATION\_MESSAGE);
177. }
178. **else**
179. JOptionPane.showMessageDialog(**new** RHDPanel(), "Failed", null, JOptionPane.INFORMATION\_MESSAGE);
180. con.close() ;
181. }
182. **catch**(Exception e) {
183. JOptionPane.showMessageDialog(**new** RHDPanel(), "Failed", null, JOptionPane.INFORMATION\_MESSAGE);
184. }
185. }
186. }
187. package ProjectCode;
188. import java.sql.\*;
190. **public** **class** Derbyexample {
191. **public** **static** **void** main(String[] args) throws Exception {
192. String jdbcURL = "jdbc:derby:data;create=false";
194. Connection connection =DriverManager.getConnection(jdbcURL);
195. System.out.println("create successfully");
197. String sqlcreat1 = "Create Table roomInf(roomType varchar(10), roomNum int, state varchar(10),price double)";
198. String sqladd1 = "Insert into roomInf(roomType, roomNum, state, price) values ('Plum', 101, 'available',100.00)";
200. String sqladd2 = "Insert into roomInf(roomType, roomNum, state, price) values ('Orchid', 102, 'reserved',100.00)";
201. String sqladd3 = "Insert into roomInf(roomType, roomNum, state, price) values ('Bamboo', 103, 'available',100.00)";
202. String sqladd4 = "Insert into roomInf(roomType, roomNum, state, price) values ('Banyan', 104, 'available',100.00)";
203. String sqladd5 = "Insert into roomInf(roomType, roomNum, state, price) values ('Maple', 105, 'available',100.00)";
204. //String sqladd6 = "Insert into roomInf(roomType, roomNum, state, price) values ('Banyan', 104, 'available',100.00)";
205. //String sql = "Create Table bookingMessage(name char(10), roomNum int, cover int,phone char(10),checkInTime char(10),leaveTime char(10),type char(10))";
206. //String sql = "Insert into bookingMessage ";
207. //String sql = "Drop Table bookingMessage";
208. //String sql = "Drop Table roomInf";
209. String sqlcreat2 = "Create Table bookingMessage(name varchar(15), roomNum int, gender varchar(10),ID varchar(10),ArriveTime varchar(15), roomType varchar(10))";
211. String sqladd6 = "INSERT INTO bookingMessage  (name, roomNum,gender,ID,ArriveTime,roomType) VALUES ('Meili', 100,'famale','123456','12.03','lan')";
212. Statement statement = connection.createStatement();
213. statement.execute(sqlcreat1);
214. statement.execute(sqlcreat2);
215. statement.executeUpdate(sqladd1);
216. statement.executeUpdate(sqladd2);
217. statement.executeUpdate(sqladd3);
218. statement.executeUpdate(sqladd4);
219. statement.executeUpdate(sqladd5);
220. statement.executeUpdate(sqladd6);
221. /\*\*
223. String sqldelete1 = "Drop Table roomInf";
224. String sqldelete2 = "Drop Table bookingMessage";
225. Statement statement = connection.createStatement();
226. statement.executeUpdate(sqldelete1);
227. statement.executeUpdate(sqldelete2); \*\*/
229. }
230. }
231. package ProjectCode;
232. import java.sql.Connection;
233. import java.sql.\*;
234. import java.util.ArrayList;
235. import java.util.Vector;
237. **public** **class** Display {          //Show room & user information
238. ArrayList<People> customers ;
239. ArrayList<Room> rooms ;
240. **public** **void** setCustomer() {     //Retrieve user's information from database
241. customers = **new** ArrayList<People>() ;
242. **try** {   Class.forName("org.apache.derby.jdbc.EmbeddedDriver") ; }
243. **catch**(Exception ee) {}
244. **try** {
245. Connection con = DriverManager.getConnection("jdbc:derby:data;create=false") ;
246. Statement sql = con.createStatement() ;
247. ResultSet rs = sql.executeQuery("SELECT \* FROM bookingMessage ") ;
248. **while**(rs.next()) {
249. String name = rs.getString(1) ;
250. **int** roomNum = rs.getInt(2) ;
251. String gender = rs.getString(3) ;
252. String ID = rs.getString(4) ;
253. String ArriveTime = rs.getString(5) ;
254. //String leaveTime = rs.getString(6) ;
255. String type = rs.getString(6) ;
256. People c = **new** Students(ArriveTime, type, ID, name, gender) ;
258. customers.add(c) ;
259. }
260. con.close() ;
261. }
262. **catch**(Exception ee) {}
263. }
264. **void** setRoom() {            //Retrieve room information from database
265. rooms = **new** ArrayList<Room>() ;
266. **try** {   Class.forName("org.apache.derby.jdbc.EmbeddedDriver") ; }
267. **catch**(Exception ee) {}
268. **try** {
269. Connection con = DriverManager.getConnection("jdbc:derby:data;create=false") ;
270. Statement sql = con.createStatement() ;
271. ResultSet rs = sql.executeQuery("SELECT \* FROM roomInf ") ;
272. **while**(rs.next()) {
273. String roomType = rs.getString(1) ;
274. **int** roomNum = rs.getInt(2) ;
275. String state = rs.getString(3) ;
276. **double** price = rs.getDouble(4) ;
277. Room r = **new** Doom(roomType, roomNum, state, price) ;
278. rooms.add(r) ;
279. }
280. con.close() ;
281. }
282. **catch**(Exception ee) {}
283. }
284. **int** getRoomLength() {               //Get room amount
285. **return** rooms.size() ;
286. }
287. **int** getCustomerLength() {           //Get user amount
288. **return** customers.size() ;
289. }
290. Room getRoom(**int** i) {               //Get room info
291. **return** rooms.get(i) ;
292. }
293. People getCustomer(**int** i) {         //Get user info
294. **return** customers.get(i) ;
295. }
296. }
298. package ProjectCode;


302. **public** **class** Doom implements Room {
303. String type ;
304. String state ;
305. **int** roomNum ;
306. **double** price ;
307. Doom() {}
308. Doom(String type, **int** roomNum, String state, **double** price) {
309. **this**.type = type ;
310. **this**.roomNum = roomNum ;
311. **this**.state = state ;
312. **this**.price = price ;
313. }
314. **public** **void** setState(String state) {    //ä¿®æ”¹æˆ¿é—´çŠ¶æ€�
315. **this**.state = state ;
316. }
317. **public** **double** getPrice() {              //è¿”å›žä»·æ ¼
318. **return** price ;
319. }
320. **public** **int** getRoomNum() {               //è¿”å›žæˆ¿å�·
321. **return** roomNum ;
322. }
323. **public** String getType() {               //è¿”å›žæˆ¿é—´ç±»åž‹
324. **return** type ;
325. }
326. **public** String getState() {              //è¿”å›žæˆ¿é—´çŠ¶æ€�
327. **return** state ;
328. }
329. }
330. package ProjectCode;
332. **public** **class** Faculty implements People {                //æˆ¿å®¢ç±»
333. String ArriveTime ;         //å…¥ä½�æ—¶é—´
334. String roomType ;               //æˆ¿é—´åž‹å�·
335. String ID ;                 //ç”µè¯�å�·ç �
336. String name ;                   //å��å­—
337. String gender ;                 //äººæ•°
338. //int roomNum ;                 //æˆ¿é—´å�·
339. Faculty(String ArriveTime, String roomType, String ID, String name, String gender) {
340. **this**.gender = gender ;
341. //this.roomNum = roomNum ;
342. **this**.ArriveTime = ArriveTime ;
344. **this**.ID = ID  ;
345. **this**.name = name  ;
346. **this**.roomType = roomType ;
347. }

350. **public** String getName() {
351. **return** **this**.name ;
352. }
354. @Override
355. **public** String getGender() {
356. // TODO Auto-generated method stub
357. **return** **this**.gender;
358. }
359. @Override
360. **public** String getID() {
361. // TODO Auto-generated method stub
362. **return** **this**.ID;
363. }
364. @Override
365. **public** String getArriveTime() {
367. **return** **this**.ArriveTime;
368. }
369. @Override
370. **public** **void** setGender(String gender) {
371. **this**.gender = gender;
373. }
374. @Override
375. **public** **void** setArriveTime(String ArriveTime) {
376. **this**.ArriveTime = ArriveTime;
378. }
379. @Override
380. **public** **void** setID(String ID) {
381. **this**.ID = ID;
383. }
384. @Override
385. **public** **void** setName(String name) {
386. **this**.name = name;
388. }

391. @Override
392. **public** String getRoomType() {
394. **return** **this**.roomType;
395. }

398. @Override
399. **public** **void** setRoomType(String roomType) {
400. **this**.roomType = roomType;
402. }
403. }
404. package ProjectCode;
405. import java.sql.\*;
407. **public** **class** Derbyexample {
408. **public** **static** **void** main(String[] args) throws Exception {
409. String jdbcURL = "jdbc:derby:data;create=false";
411. Connection connection =DriverManager.getConnection(jdbcURL);
412. System.out.println("create successfully");
414. String sqlcreat1 = "Create Table roomInf(roomType varchar(10), roomNum int, state varchar(10),price double)";
415. String sqladd1 = "Insert into roomInf(roomType, roomNum, state, price) values ('Plum', 101, 'available',100.00)";
417. String sqladd2 = "Insert into roomInf(roomType, roomNum, state, price) values ('Orchid', 102, 'reserved',100.00)";
418. String sqladd3 = "Insert into roomInf(roomType, roomNum, state, price) values ('Bamboo', 103, 'available',100.00)";
419. String sqladd4 = "Insert into roomInf(roomType, roomNum, state, price) values ('Banyan', 104, 'available',100.00)";
420. String sqladd5 = "Insert into roomInf(roomType, roomNum, state, price) values ('Maple', 105, 'available',100.00)";
421. //String sqladd6 = "Insert into roomInf(roomType, roomNum, state, price) values ('Banyan', 104, 'available',100.00)";
422. //String sql = "Create Table bookingMessage(name char(10), roomNum int, cover int,phone char(10),checkInTime char(10),leaveTime char(10),type char(10))";
423. //String sql = "Insert into bookingMessage ";
424. //String sql = "Drop Table bookingMessage";
425. //String sql = "Drop Table roomInf";
426. String sqlcreat2 = "Create Table bookingMessage(name varchar(15), roomNum int, gender varchar(10),ID varchar(10),ArriveTime varchar(15), roomType varchar(10))";
428. String sqladd6 = "INSERT INTO bookingMessage  (name, roomNum,gender,ID,ArriveTime,roomType) VALUES ('Meili', 100,'famale','123456','12.03','lan')";
429. Statement statement = connection.createStatement();
430. statement.execute(sqlcreat1);
431. statement.execute(sqlcreat2);
432. statement.executeUpdate(sqladd1);
433. statement.executeUpdate(sqladd2);
434. statement.executeUpdate(sqladd3);
435. statement.executeUpdate(sqladd4);
436. statement.executeUpdate(sqladd5);
437. statement.executeUpdate(sqladd6);
438. /\*\*
440. String sqldelete1 = "Drop Table roomInf";
441. String sqldelete2 = "Drop Table bookingMessage";
442. Statement statement = connection.createStatement();
443. statement.executeUpdate(sqldelete1);
444. statement.executeUpdate(sqldelete2); \*\*/
446. }
447. }
448. package ProjectCode;
449. import java.sql.Connection;
450. import java.sql.\*;
451. import java.util.ArrayList;
452. import java.util.Vector;
454. **public** **class** Display {          //Show room & user information
455. ArrayList<People> customers ;
456. ArrayList<Room> rooms ;
457. **public** **void** setCustomer() {     //Retrieve user's information from database
458. customers = **new** ArrayList<People>() ;
459. **try** {   Class.forName("org.apache.derby.jdbc.EmbeddedDriver") ; }
460. **catch**(Exception ee) {}
461. **try** {
462. Connection con = DriverManager.getConnection("jdbc:derby:data;create=false") ;
463. Statement sql = con.createStatement() ;
464. ResultSet rs = sql.executeQuery("SELECT \* FROM bookingMessage ") ;
465. **while**(rs.next()) {
466. String name = rs.getString(1) ;
467. **int** roomNum = rs.getInt(2) ;
468. String gender = rs.getString(3) ;
469. String ID = rs.getString(4) ;
470. String ArriveTime = rs.getString(5) ;
471. //String leaveTime = rs.getString(6) ;
472. String type = rs.getString(6) ;
473. People c = **new** Students(ArriveTime, type, ID, name, gender) ;
475. customers.add(c) ;
476. }
477. con.close() ;
478. }
479. **catch**(Exception ee) {}
480. }
481. **void** setRoom() {            //Retrieve room information from database
482. rooms = **new** ArrayList<Room>() ;
483. **try** {   Class.forName("org.apache.derby.jdbc.EmbeddedDriver") ; }
484. **catch**(Exception ee) {}
485. **try** {
486. Connection con = DriverManager.getConnection("jdbc:derby:data;create=false") ;
487. Statement sql = con.createStatement() ;
488. ResultSet rs = sql.executeQuery("SELECT \* FROM roomInf ") ;
489. **while**(rs.next()) {
490. String roomType = rs.getString(1) ;
491. **int** roomNum = rs.getInt(2) ;
492. String state = rs.getString(3) ;
493. **double** price = rs.getDouble(4) ;
494. Room r = **new** Doom(roomType, roomNum, state, price) ;
495. rooms.add(r) ;
496. }
497. con.close() ;
498. }
499. **catch**(Exception ee) {}
500. }
501. **int** getRoomLength() {               //Get room amount
502. **return** rooms.size() ;
503. }
504. **int** getCustomerLength() {           //Get user amount
505. **return** customers.size() ;
506. }
507. Room getRoom(**int** i) {               //Get room info
508. **return** rooms.get(i) ;
509. }
510. People getCustomer(**int** i) {         //Get user info
511. **return** customers.get(i) ;
512. }
513. }
515. package ProjectCode;


519. **public** **class** Doom implements Room {
520. String type ;
521. String state ;
522. **int** roomNum ;
523. **double** price ;
524. Doom() {}
525. Doom(String type, **int** roomNum, String state, **double** price) {
526. **this**.type = type ;
527. **this**.roomNum = roomNum ;
528. **this**.state = state ;
529. **this**.price = price ;
530. }
531. **public** **void** setState(String state)
532. **this**.state = state ;
533. }
534. **public** **double** getPrice() {
535. **return** price ;
536. }
537. **public** **int** getRoomNum() {
538. **return** roomNum ;
539. }
540. **public** String getType() {
541. **return** type ;
542. }
543. **public** String getState() {
544. **return** state ;
545. }
546. }
547. package ProjectCode;
549. **public** **class** Faculty implements People {
550. String ArriveTime ;
551. String roomType ;
552. String ID ;
553. String name ;
554. String gender ;
555. //int roomNum ;
556. Faculty(String ArriveTime, String roomType, String ID, String name, String gender) {
557. **this**.gender = gender ;
558. //this.roomNum = roomNum ;
559. **this**.ArriveTime = ArriveTime ;
561. **this**.ID = ID  ;
562. **this**.name = name  ;
563. **this**.roomType = roomType ;
564. }

567. **public** String getName() {
568. **return** **this**.name ;
569. }
571. @Override
572. **public** String getGender() {
573. // TODO Auto-generated method stub
574. **return** **this**.gender;
575. }
576. @Override
577. **public** String getID() {
578. // TODO Auto-generated method stub
579. **return** **this**.ID;
580. }
581. @Override
582. **public** String getArriveTime() {
584. **return** **this**.ArriveTime;
585. }
586. @Override
587. **public** **void** setGender(String gender) {
588. **this**.gender = gender;
590. }
591. @Override
592. **public** **void** setArriveTime(String ArriveTime) {
593. **this**.ArriveTime = ArriveTime;
595. }
596. @Override
597. **public** **void** setID(String ID) {
598. **this**.ID = ID;
600. }
601. @Override
602. **public** **void** setName(String name) {
603. **this**.name = name;
605. }

608. @Override
609. **public** String getRoomType() {
611. **return** **this**.roomType;
612. }

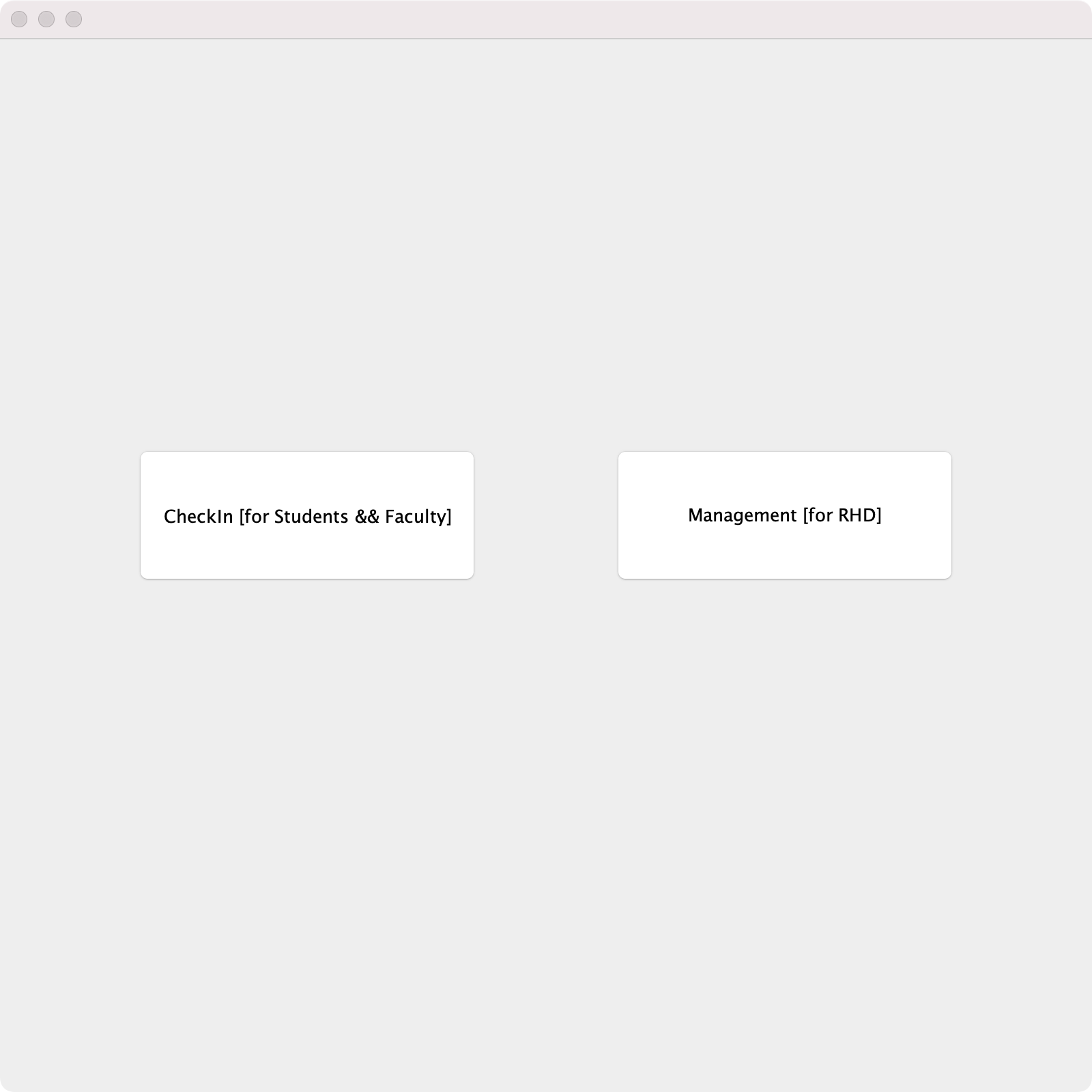
615. @Override
616. **public** **void** setRoomType(String roomType) {
617. **this**.roomType = roomType;
619. }
620. }
621. package ProjectCode;
623. **public** interface People {
624. String getGender();
625. //int getRoomNum();
626. String getName() ;
627. String getID() ;
628. String getArriveTime() ;
629. String getRoomType();
631. **void** setGender(String gender) ;
632. //void setRoomNum(int roomNum);
633. **void** setArriveTime(String ArriveTime);
634. **void** setID(String phone) ;
635. **void** setName(String name) ;
636. **void** setRoomType(String roomType);
637. }
638. **package** ProjectCode;
640. **import** java.awt.event.\*;
641. **import** java.sql.\*;
642. **import** javax.swing.\*;
644. @SuppressWarnings("serial")
645. **public** **class** RHDPanel **extends** JPanel **implements** ActionListener{
646. JLabel label ;
647. JTextField ID ;
648. JButton arrive ;                //Record user's arrival
649. JButton cancel ;                //Reject application
650. JButton transfer ;              //Transfer to Quarantine area
651. JTextArea customerMessage ;     ///Show user message area
652. JTextArea roomMessage ;         //Show room message area
653. Display display ;
654. **public** RHDPanel() {
655. setLayout(**null**) ;
656. setBounds(0, 0, 800, 800) ;
657. setVisible(**true**) ;
658. label = **new** JLabel("Please enter your ID") ;
659. label.setBounds(100, 60, 100, 50) ;
660. ID = **new** JTextField(30) ;
661. ID.setBounds(210, 70, 400, 30) ;
662. arrive = **new** JButton("Approve") ;
663. arrive.setBounds(200, 120, 100, 45) ;
664. arrive.addActionListener(**this**) ;
665. cancel = **new** JButton("Reject") ;
666. cancel.setBounds(350, 120, 100, 45) ;
667. cancel.addActionListener(**this**) ;
668. transfer = **new** JButton("Modify") ;
669. transfer.setBounds(500, 120, 100, 45) ;
670. transfer.addActionListener(**this**) ;
671. customerMessage  = **new** JTextArea(12, 20) ;
672. customerMessage.setBounds(150, 200, 500, 200) ;
673. roomMessage = **new** JTextArea(12, 20) ;
674. roomMessage.setBounds(150, 450, 500, 200) ;
675. add(label) ;
676. add(ID) ;
677. add(arrive) ;
678. add(cancel) ;
679. add(transfer) ;
680. add(customerMessage) ;
681. add(roomMessage) ;
682. setCustomerInf() ;
683. setRoomInf() ;
684. }
685. **void** setCustomerInf() {             //Show all applications
686. display = **new** Display() ;
687. display.setCustomer() ;             //Read information from database
688. customerMessage.append("Name        Room type         Gender        ID          Arrive time \n") ;
689. **for**(**int** i=0; i<display.getCustomerLength(); i++) {
690. People customerInf = display.getCustomer(i) ;
691. customerMessage.append(customerInf.getName()+"       "+customerInf.getRoomType()+"        "
692. +customerInf.getGender()+"               "+customerInf.getID()+"               "
693. +customerInf.getArriveTime());//+"           ";
694. //+customerInf.getLeaveTime()) ;
695. customerMessage.append("\n") ;
696. }
697. }
698. **void** setRoomInf() {             //Show all room information
699. display = **new** Display() ;
700. display.setRoom() ;             //Read information from database
701. roomMessage.append(" Roomtype  Room number   price   Room state\n") ;
702. **for**(**int** i=0; i<display.getRoomLength(); i++) {
703. Room roomInf = display.getRoom(i) ;
704. roomMessage.append(roomInf.getType()+"    "+roomInf.getRoomNum()+"       "
705. +"                "+roomInf.getState()) ;
706. roomMessage.append("\n") ;
707. }
708. }
709. **public** **void** actionPerformed(ActionEvent e) {
710. **if**(e.getSource() == arrive) {           //Mark as arrive
711. SignIn signIn = **new** SignIn(ID.getText()) ;
712. customerMessage.setText("");        //Update information¯
713. roomMessage.setText("") ;
714. setCustomerInf() ;
715. setRoomInf() ;
716. }
717. **if**(e.getSource() == cancel) {           //Cancel application
718. Cancel cancel = **new** Cancel(ID.getText()) ;
719. customerMessage.setText("");        //Update information
720. roomMessage.setText("") ;
721. setCustomerInf() ;
722. setRoomInf() ;
723. }
724. **if**(e.getSource() == transfer) {         //Transfer to quarantine area
725. **int** newRoom = Integer.parseInt(JOptionPane.showInputDialog(**this**,
726. "Please enter the modified room number ", **null**, JOptionPane.INFORMATION\_MESSAGE)) ;
727. Change change = **new** Change(ID.getText(), newRoom) ;
728. customerMessage.setText("");        //Update information
729. roomMessage.setText("") ;
730. setCustomerInf() ;
731. setRoomInf() ;
732. }
733. }
734. }
735. **package** ProjectCode;
737. **import** java.awt.event.\*;
738. **import** java.sql.\*;
739. **import** javax.swing.\*;
741. @SuppressWarnings("serial")
742. **public** **class** RHDPanel **extends** JPanel **implements** ActionListener{
743. JLabel label ;
744. JTextField ID ;
745. JButton arrive ;                //Record user's arrival
746. JButton cancel ;                //Reject application
747. JButton transfer ;              //Transfer to Quarantine area
748. JTextArea customerMessage ;     ///Show user message area
749. JTextArea roomMessage ;         //Show room message area
750. Display display ;
751. **public** RHDPanel() {
752. setLayout(**null**) ;
753. setBounds(0, 0, 800, 800) ;
754. setVisible(**true**) ;
755. label = **new** JLabel("Please enter your ID") ;
756. label.setBounds(100, 60, 100, 50) ;
757. ID = **new** JTextField(30) ;
758. ID.setBounds(210, 70, 400, 30) ;
759. arrive = **new** JButton("Approve") ;
760. arrive.setBounds(200, 120, 100, 45) ;
761. arrive.addActionListener(**this**) ;
762. cancel = **new** JButton("Reject") ;
763. cancel.setBounds(350, 120, 100, 45) ;
764. cancel.addActionListener(**this**) ;
765. transfer = **new** JButton("Modify") ;
766. transfer.setBounds(500, 120, 100, 45) ;
767. transfer.addActionListener(**this**) ;
768. customerMessage  = **new** JTextArea(12, 20) ;
769. customerMessage.setBounds(150, 200, 500, 200) ;
770. roomMessage = **new** JTextArea(12, 20) ;
771. roomMessage.setBounds(150, 450, 500, 200) ;
772. add(label) ;
773. add(ID) ;
774. add(arrive) ;
775. add(cancel) ;
776. add(transfer) ;
777. add(customerMessage) ;
778. add(roomMessage) ;
779. setCustomerInf() ;
780. setRoomInf() ;
781. }
782. **void** setCustomerInf() {             //Show all applications
783. display = **new** Display() ;
784. display.setCustomer() ;             //Read information from database
785. customerMessage.append("Name        Room type         Gender        ID          Arrive time \n") ;
786. **for**(**int** i=0; i<display.getCustomerLength(); i++) {
787. People customerInf = display.getCustomer(i) ;
788. customerMessage.append(customerInf.getName()+"       "+customerInf.getRoomType()+"        "
789. +customerInf.getGender()+"               "+customerInf.getID()+"               "
790. +customerInf.getArriveTime());//+"           ";
791. //+customerInf.getLeaveTime()) ;
792. customerMessage.append("\n") ;
793. }
794. }
795. **void** setRoomInf() {             //Show all room information
796. display = **new** Display() ;
797. display.setRoom() ;             //Read information from database
798. roomMessage.append(" Roomtype  Room number   price   Room state\n") ;
799. **for**(**int** i=0; i<display.getRoomLength(); i++) {
800. Room roomInf = display.getRoom(i) ;
801. roomMessage.append(roomInf.getType()+"    "+roomInf.getRoomNum()+"       "
802. +"                "+roomInf.getState()) ;
803. roomMessage.append("\n") ;
804. }
805. }
806. **public** **void** actionPerformed(ActionEvent e) {
807. **if**(e.getSource() == arrive) {           //Mark as arrive
808. SignIn signIn = **new** SignIn(ID.getText()) ;
809. customerMessage.setText("");        //Update information¯
810. roomMessage.setText("") ;
811. setCustomerInf() ;
812. setRoomInf() ;
813. }
814. **if**(e.getSource() == cancel) {           //Cancel application
815. Cancel cancel = **new** Cancel(ID.getText()) ;
816. customerMessage.setText("");        //Update information
817. roomMessage.setText("") ;
818. setCustomerInf() ;
819. setRoomInf() ;
820. }
821. **if**(e.getSource() == transfer) {         //Transfer to quarantine area
822. **int** newRoom = Integer.parseInt(JOptionPane.showInputDialog(**this**,
823. "Please enter the modified room number ", **null**, JOptionPane.INFORMATION\_MESSAGE)) ;
824. Change change = **new** Change(ID.getText(), newRoom) ;
825. customerMessage.setText("");        //Update information
826. roomMessage.setText("") ;
827. setCustomerInf() ;
828. setRoomInf() ;
829. }
830. }
831. }
832. **package** ProjectCode;
834. **import** java.awt.event.\*;
835. **import** java.sql.\*;
836. **import** javax.swing.\*;
838. **public** **class** SignIn {
839. Connection con ;
840. Statement sql ;
841. ResultSet rs ;
842. SignIn(String phone) {
843. sign(phone) ;
844. }
845. **void** sign(String ID) {      //update status
846. **try** {   Class.forName("org.apache.derby.jdbc.EmbeddedDriver") ; }
847. **catch**(Exception e) {}
848. **try** {
849. con = DriverManager.getConnection("jdbc:derby:data;create=false") ;
850. sql = con.createStatement() ;
852. //System.out.print("Id is : "+ID);
853. rs = sql.executeQuery("SELECT \* FROM bookingMessage WHERE ID='"+ID+"'") ;//search by id
854. //System.out.print("Id is : "+ID);
856. rs.next() ;             //jump to next, which is the search result
857. //System.out.print("enter0");
859. **int** roomNum = rs.getInt(2) ;        //get room number
860. //System.out.print("enter1");
861. sql.executeLargeUpdate("UPDATE roomInf SET state='arrived' WHERE roomNum="+roomNum) ;   //update room status
863. //System.out.print("enter2");
865. JOptionPane.showMessageDialog(**new** RHDPanel(), "Arrive success", **null**, JOptionPane.INFORMATION\_MESSAGE);
866. con.close() ;
867. }
868. **catch**(Exception e) {
869. JOptionPane.showMessageDialog(**new** RHDPanel(), "Arrive Fail", **null**, JOptionPane.INFORMATION\_MESSAGE);
870. }
871. }
872. }
873. **package** ProjectCode;
874. **import** java.awt.event.\*;
875. **import** java.sql.\*;
876. **import** javax.swing.\*;
878. **public** **class** StudentFill **extends** OrderPanel **implements** ActionListener{      //record student application
879. **public** **void** actionPerformed(ActionEvent e) {
880. **if**(e.getSource() == button) {
881. **try** {   Class.forName("org.apache.derby.jdbc.EmbeddedDriver") ; }
882. **catch**(Exception ee) {}
883. **try** {
884. Connection con = DriverManager.getConnection("jdbc:derby:data;create=false") ;
885. Statement sql = con.createStatement() ;
886. //System.out.println("room type is: "+roomType.getText());
887. ResultSet rs = sql.executeQuery("SELECT \* FROM roomInf WHERE roomType='"+roomType.getText()+"'") ;//search by room type
888. //System.out.println("room type is: "+roomType.getText());
890. **boolean** flag = **true** ;           //mark as success
891. **while**(rs.next()) {
892. String type = rs.getString(1) ;
893. **int** number = rs.getInt(2) ;
894. String state = rs.getString(3) ;
896. **if**(state.equals("available")) {             //check the target room is available or not
898. sql.executeLargeUpdate("UPDATE roomInf SET state='reserved' WHERE roomNum="+number) ;       //update room status

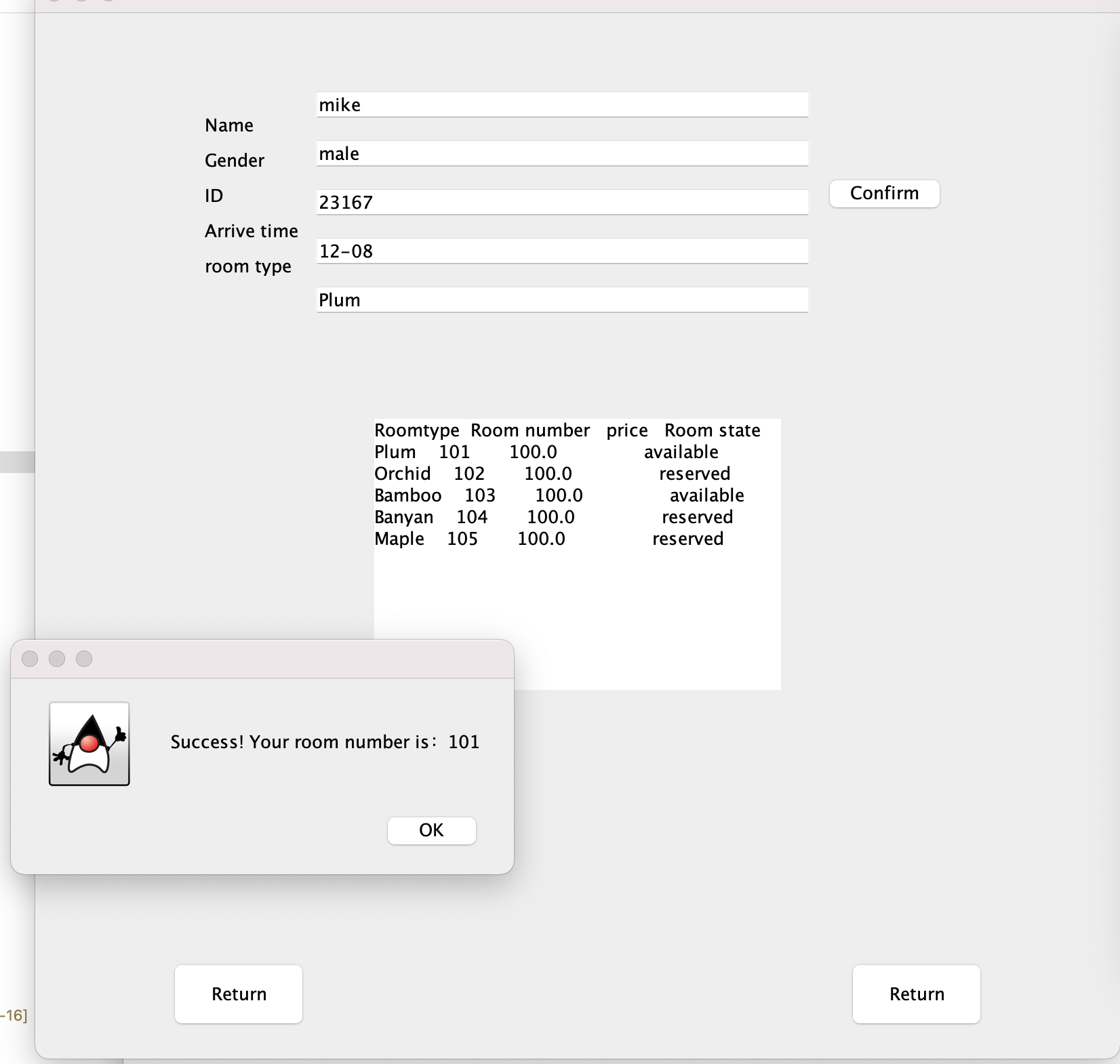
901. sql.executeUpdate("INSERT INTO bookingMessage VALUES ('"+name.getText()+
902. "', "+number+", '"+gender.getText()+"', '"+ID.getText()+"', '"+
903. ArriveTime.getText()+"', '"+
904. type+"')") ;                //store in database
906. System.out.println("enter2");
908. JOptionPane.showMessageDialog(**this**, "Success! Your room number is: "+number+" ",**null**, JOptionPane.INFORMATION\_MESSAGE);
909. flag = **false** ;
910. **break** ;
911. }
912. }
913. **if**(flag)
914. JOptionPane.showMessageDialog(**this**, "Fail!",**null**, JOptionPane.INFORMATION\_MESSAGE);
915. con.close() ;
916. }
917. **catch**(Exception ee) {}
918. }
919. }
920. }
921. **package** ProjectCode;
923. **public** **class** Students **implements** People {
924. String ArriveTime ;
925. String roomType ;
926. String ID ;
927. String name ;
928. String gender ;
930. Students(String ArriveTime, String roomType, String ID, String name, String gender) {
931. **this**.gender = gender ;
932. **this**.ArriveTime = ArriveTime ;
933. **this**.ID = ID  ;
934. **this**.name = name  ;
935. **this**.roomType = roomType ;
936. }

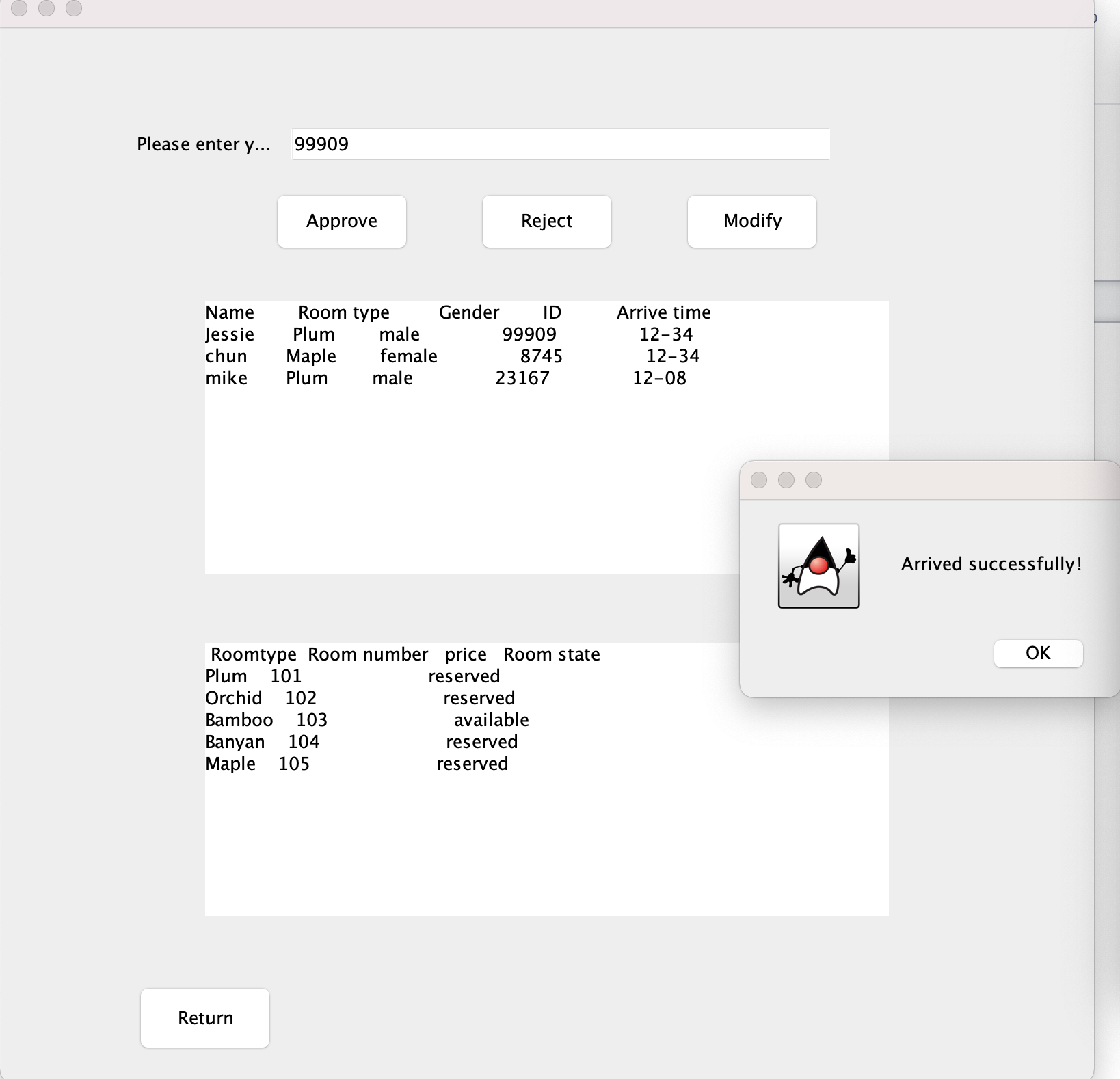
939. **public** String getName() {
940. **return** **this**.name ;
941. }
943. @Override
944. **public** String getGender() {
945. // TODO Auto-generated method stub
946. **return** **this**.gender;
947. }
948. @Override
949. **public** String getID() {
950. // TODO Auto-generated method stub
951. **return** **this**.ID;
952. }
953. @Override
954. **public** String getArriveTime() {
956. **return** **this**.ArriveTime;
957. }
958. @Override
959. **public** **void** setGender(String gender) {
960. **this**.gender = gender;
962. }
963. @Override
964. **public** **void** setArriveTime(String ArriveTime) {
965. **this**.ArriveTime = ArriveTime;
967. }
968. @Override
969. **public** **void** setID(String ID) {
970. **this**.ID = ID;
972. }
973. @Override
974. **public** **void** setName(String name) {
975. **this**.name = name;
977. }

980. @Override
981. **public** String getRoomType() {
983. **return** **this**.roomType;
984. }

987. @Override
988. **public** **void** setRoomType(String roomType) {
989. **this**.roomType = roomType;
991. }
992. }
993. **package** ProjectCode;
995. **import** java.awt.event.\*;
996. **import** java.sql.\*;
997. **import** javax.swing.\*;
999. @SuppressWarnings("serial")
1000. **public** **class** SystemFrame **extends** JFrame **implements** ActionListener {
1001. JButton bookingFunction ;           //register (book) function
1002. JButton arriveFunction ;            //mark as arrive function
1003. JButton changeFunction ;            //change function
1004. JButton back ;                      //back button
1005. JPanel home ;                       //home page
1006. RegisterPanel registerPanel ;       //student register panel
1007. RHDPanel rhdPanel ;                 //RHD panel
1008. **int** flag = 0 ;                      //mark which panel is used
1009. SystemFrame() {
1010. setLayout(**null**) ;
1011. setBounds(300, 100, 800, 800);
1012. setVisible(**true**) ;
1013. setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE) ;
1014. bookingFunction = **new** JButton("CheckIn [for Students & Faculty]") ;
1015. bookingFunction.addActionListener(**this**) ;
1016. bookingFunction.setBounds(100, 300, 250, 100);
1017. arriveFunction = **new** JButton("Management [for RHD]") ;
1018. arriveFunction.addActionListener(**this**) ;
1019. arriveFunction.setBounds(450, 300, 250, 100);
1020. back = **new** JButton("Return") ;
1021. back.setBounds(100, 700, 100, 50) ;
1022. back.addActionListener(**this**) ;
1023. back.setVisible(**false**) ;
1024. home = **new** JPanel() ;
1025. home.setLayout(**null**) ;
1026. home.setBounds(0, 0, 800, 800) ;
1027. home.add(bookingFunction) ;
1028. home.add(arriveFunction) ;
1029. add(back) ;
1030. add(home) ;
1031. }
1032. **public** **void** actionPerformed(ActionEvent e) {
1033. **if**(e.getSource() == bookingFunction) {      //click to enter the function
1034. registerPanel = **new** RegisterPanel() ;
1035. home.setVisible(**false**) ;
1036. remove(home) ;
1037. add(registerPanel) ;
1038. back.setVisible(**true**) ;
1039. flag = 1 ;
1040. }
1041. **if**(e.getSource() == arriveFunction) {       //click to enter the function
1042. rhdPanel = **new** RHDPanel() ;
1043. home.setVisible(**false**) ;
1044. remove(home) ;
1045. add(rhdPanel) ;
1046. back.setVisible(**true**) ;
1047. flag = 2 ;
1048. }
1049. **if**(e.getSource() == back) {             //back to previous page
1050. **if**(flag == 1) {
1051. registerPanel.setVisible(**false**) ;
1052. remove(registerPanel) ;             //remove text box
1053. }
1054. **if**(flag == 2) {
1055. rhdPanel.setVisible(**false**) ;        //remove RHD panel
1056. remove(rhdPanel) ;
1057. }
1058. home.setVisible(**true**) ;
1059. add(home) ;                         //show buttons again
1060. back.setVisible(**false**) ;
1061. }
1062. }
1063. }







**VII. Conclusions**

This return-campus management system implements two main functions, that is using database to record students and faculty’s campus returning information, modifying room and resident information to track and record people’s campus returning information.

This program using Swing&AWT to develop a graphical interface(to display room and resident information), using Apache Derby to construct the database to achieve managing the data about people and room successfully and easily just through controlling different function buttons.

The code of this return-campus management system also implements four decoding method, that is Singleton Method,Factory Method,Builder Method and Observer Method to create instances, build connections between objects.

Finally, this return-campus management system achieve to simply simulate WKU’s return campus management system and enables administrators to process return-campus applications, arrange dormitory distribution, and manage information database which can be completed by simple button operations easily.

**References**

*Introduction to Derby*. Getting started with Derby. (2004). Retrieved December 13, 2021, from https://db.apache.org/derby/docs/10.15/getstart/index.html.

The Java™ Tutorials. (n.d.). *Lesson: JDBC basics*. Lesson: JDBC Basics (The Java™ Tutorials > JDBC Database Access). Retrieved December 13, 2021, from https://docs.oracle.com/javase/tutorial/jdbc/basics/index.html.

*Object-oriented programming concepts*. Lesson: Object-Oriented Programming Concepts (The Java™ Tutorials > Learning the Java Language). (n.d.). Retrieved December 13, 2021, from https://docs.oracle.com/javase/tutorial/java/concepts/.

**Appendix:**

**Contribution of Group Members**

Wang Junfeng 1129618 & Liu Meili 1192738

* System framework design
* Graphical User Interface design
* Derby database access and connection using JDBC
* Code building and debugging

Wu Kangmin 1162949

* Project requirements & function planning
* Class hierarchy design
* UML diagram building