1. 로그인&로그아웃&회원가입 = 윤원택

```
525         signInBtn.addMouseListener(
526                 new SignInEventHandler(
527                         (CardLayout)userCards.getLayout(),
528                         userCards,
529                         adminPageBtn,
530                         phoneTextField,
531                         passwordField,
532                         userRepo));
533 
534         logOffBtn.addMouseListener(
535                 new SignOffEventHandler(
536                         (CardLayout)userCards.getLayout(),
537                         userCards,
538                         adminPageBtn,
539                         phoneTextField,
540                         passwordField,
541                         modelOrdering,
542                         userRepo));
543 
544         signUpBtn.addMouseListener(this);
```

회원가입 버튼(signUpBtn) 로그인 버튼(signInBtn) 로그아웃 버튼(signOffBtn)에 mouse listener를 추가하여, mouse관련 event를 처리하도록 하였습니다.

회원가입 버튼 클릭 이벤트 발생시, 메인화면에서, CardLayout 의 SignUpPageCard 패널 카드를 보여주도록 했습니다. 이 패널의 디자인 및 이벤트는 SignUpPageCard.java에 정의하였습니다.

회원가입 버튼 클릭시 SignUpPageCard는 유저의 개인정보를 입력받는데, 비밀번호String 값을 Hash 알고리즘으로 암호화하여 데이터를 저장합니다. 다음 링크에서 소스코드를 참고하였습니다 :* https://howtodoinjava.com/security/how-to-generate-secure-password-hash-md5-sha-pbkdf2-bcrypt-examples/

```
16  public class SignInEventHandler extends MouseAdapter {
```

```
36      @Override
37      public void mousePressed(MouseEvent e) {
```

로그인 로그아웃 이벤트 처리는, 각각의 파일(SignInEventHandler.java, SignOffEventHandler.java)로 구현하였습니다.

로그인 할때도, 입력한 비밀번호값을 Hash알고리즘으로 암호화 하고, 회원가입 시 이미 암호화 하여 저장된 데이터와 일치여부를 확인합니다. 로그인에 성공하면, 핸드폰과 비밀번호 입력 필드을 setEditable(false)로 바꿔줍니다.

회원가입 예외상황 처리 (Exception Handling)

1. 로그인된 사용자가 있는경우

2. 비밀번호가 4자리 이상이 아닌경우

3. 회원정보 필드값이 하나라도 빠진경우

로그인 예외상황 처리

1. 사용자가 이미 로그인 되어있는경우

2. 핸드폰번호 또는 비밀번호가 누락된 경우

3. 핸드폰번호 또는 비밀번호가 불일치

2. 주문하기 (2명) = 문준석, 최호준

3개의 메뉴가 있으므로, CardLayout에 3장의 패널을 넣었습니다. 각각의 카드 패널에는 DefaultTableModel의 데이터를 넣은 JTable이 있습니다.

```
599     private TableModel createDefaultTableModel() {
600         String[] colNames = {"카테고리", "메뉴번호", "메뉴이름", "가격"};
601 
602         return new DefaultTableModel(colNames, 0) {
603             private static final long serialVersionUID = 1L;
604             String[] colNames = {"카테고리", "메뉴번호", "메뉴이름", "가격"};
605             @Override
606             public String getColumnName(int column) {
607                 return colNames[column];
608             }
609             @Override
610             public boolean isCellEditable(int row, int column) {
611                return false;
612             }
613         };
614     }
```

DefaultTableModel 객체를 생성시 DefaultTableModel inner 클래스 정의에서 isCellEditable 이 false를 반환하게 하여, 테이블 행값을 변경하지 못하도록 했습니다.

```
637         FoodMenu menu = userRepo.getFoodMenu();
638         List<Food> foodMenuList = null;
639         Iterator<Food> itr = null;
640         if(menu!= null)
641             foodMenuList = menu.getFoodMenuList();
642 
```

```
677         String[][] noodleList = new String[countN][colNames.length];
678         String[][] soupList = new String[countS][colNames.length];
679         String[][] riceList = new String[countR][colNames.length];
680 
```

```
694         for(int i =0;  i<noodleList.length; i++)  modelN.addRow(noodleList[i]);
695         for(int i =0;  i<soupList.length; i++)  modelS.addRow(soupList[i]);
696         for(int i =0;  i<riceList.length; i++)  modelR.addRow(riceList[i]);
```

Food 리스트를 데이터에서 가져와서 2차원 배열에 데이터를 넣고, DefaultTableModel에

이 데이터를 추가하여, 다시 JTable에 데이터를 넣었습니다. 2차원 배열대신에 Vector로 데이터를 넣는 방법도 있다하여 나중에 시도해보려고 합니다.

음식 추가버튼 (초록생 장바구니 아이콘) 클릭시 해당 음식 메뉴와, 주문 개수가 임시 데이터로(tempOrderList) 저장되고, '주문하기' 버튼 클릭시, 객체(User)의 필드값(Map<Food, Integer>orderList) 데이터에 저장합니다.

```
855         Food food = null;
856         int qty  =0;
857         for(Map.Entry<Food, Integer> entry : orderList.entrySet()) {
858             food = entry.getKey();
859             qty = entry.getValue();
860 
861             if(salesResult.get(food) != null)
862                 salesResult.put(food, salesResult.get(food) + qty);
863             else
864                 salesResult.put(food, qty);
865         }
866         ((Admin)admin).setSalesResult(salesResult);
```

이와 함께, Admin 객체의 총 매출결과 (Map<Food,Integer> salesResult) 데이터도 변경 합니다.

카드 및 현금 선택하여 주문시 CARD, CASH 등 String 값이 User의 필드에 저장되어,

이를 바탕으로 상세 결제 절차를 차후에 구현할 가능성을 열어놓았습니다.

주문 완료시, 인기메뉴 top5를 업데이트합니다. 현재 Admin의 매출(Map <Food, Integer> salesResult) 데이터의 value값에는 해당 음식 메뉴의 총 판매량이 저장되어 있습니다.

key값을 바탕으로 정렬하는 Map의 특성상, key와 value값을 뒤집어서
List<Map.Entry<Integer,Food>> sortedSales; 에 데이터를 넣었습니다.

```
1089        Map<Food, Integer> salesResult = (TreeMap<Food, Integer>)((Admin)userRepo.getAdmin()).getSalesResult();
1090        if (salesResult == null)
1091            return;
1092 
1093        List<Map.Entry<Integer, Food>> sortedSales = new ArrayList<Map.Entry<Integer, Food>>();
1094 
1095        Map.Entry<Integer, Food> newEntry = null;
1096        for(Map.Entry<Food, Integer> entry : salesResult.entrySet()) {
1097            newEntry = new AbstractMap.SimpleEntry<Integer, Food>(entry.getValue(), entry.getKey());
1098            sortedSales.add(newEntry);
1099        }
1100 
1101        Collections.sort(sortedSales, (i,j)->{
1102            return j.getKey() - i.getKey();
1103        });
1104 
```

이 리스트안에 Map.Entry<Integer, Food> 엔트리가 있고, Integer(총매출 개수) 순서로 정렬하기 위해 Collections.sort를 사용하여 상위 5위까지의 음식 메뉴와 매출개수를 보여주었습니다.

3. 주문취소, 메인화면과 객체간 연결 = 이준호

```
10  public class User implements Serializable, Comparable<User> {
11      private String username;
12      private String phone;
13      private String password;
14      private String email;
15      private String address;
16      private boolean logged; //로그인 상태 여부
17      private Map<Food, Integer> orderList;
18      private GregorianCalendar orderCreated;
19      private String recentPayMethod;
20      private int seatNo; //좌석(1~SEATS)
21      private boolean ordering;
```

간단히 객체에 대해 설명드리면, User는 음식주문 사용자를 정의하는 클래스로 Comparable 인터페이스를 상속합니다.

```
905             this.tempOrderList = new TreeMap<Food, Integer>();
```

User클래스에 있는 주문내역(Map<Food, Integer> orderList;)은 Food에 정의된 compareTo 메소드를 를 기반으로 자동정렬되는데, 이것은 TreeMap의 특성으로 가능합니다. 클래스에서 Map으로 정의한 orderList를 초기화 시에 객체 다형성을 이용하여 TreeMap으로 초기화 시켰습니다. 따라서 orderList의 정렬은 TreeMap에 정의된 방법을 따릅니다.

```
905             this.tempOrderList = new TreeMap<Food, Integer>();
```

Food 음식메뉴 또한 Comparable<Food> 인터페이스를 상속하여, 이것을 key값으로 하는 TreeMap<Food, ?> 컬렉션 사용 시, Food 클래스의 오버라이드 메소드 compareTo에 정의한 규칙에 따라 데이터가 자동 정렬 됩니다.

```
328         List<Food> foodMenuList = (ArrayList<Food>)userRepo.getFoodMenu().getFoodMenuList();
329
330         Collections.sort(foodMenuList, (i,j)->{
331             return i.compareTo(j);
332         });
```

FoodMenu는 음식메뉴 List를 정의하는 클래스로, 이 List<Food> foodMenuList; 는 Collections.sort를 이용하여 정렬하였습니다.

Admin클래스에 있는 메뉴별 매출(Map<Food,Integer> salesResult;)은 TreeMap의 자동정렬 특성과, Food의 compareTo에 정의된 규칙에 따라, 메뉴 알파벳 순 정렬 하였습니다.

```
217         List<Map.Entry<Integer, Food>> sortedSales = new ArrayList<Map.Entry<Integer, Food>>();
218
219         Map.Entry<Integer, Food> newEntry = null;
220         for(Map.Entry<Food, Integer> entry : salesResult.entrySet()) {
221             newEntry = new AbstractMap.SimpleEntry<Integer, Food>(entry.getValue(), entry.getKey());
222             sortedSales.add(newEntry);
223         }
224
225         Collections.sort(sortedSales, (i,j)->{
226             return j.getKey() - i.getKey();
227         });
```

다만 총매출 조회시에 가장 많이 팔린 개수(Integer) 기준 정렬은, Map의 Value값을 기준으로 정렬할 수 없기 때문에, List<Map.Entry<Integer, Food>> sortedSales;에 데이터를 넣고,

Collections.sort으로 내림차순 정렬 했습니다.

UserRepository는 사용자 관련 정보를 컬렉션 데이터(List, Map 등)을 이용하여 저장하고, 데이트 해주는 기능을 하고 있습니다. 데이터를 파일에서 read write하는 역할도 합니다.

따라서 메인화면 InitPageFrame에 UserRepository를 필드로 정의하고, 이벤트 발생시, 이 데이터를 매개변수로 전달하고, 참조변수는 매개변수 주소값을 전달하기 때문에, 해당 화면에서 데이터 변경시 함께 변경되는 특성을 이용했습니다.

주문취소는 주문취소버튼(휴지통 아이콘)을 누르면

```
807         Food food = new Food(menuCategory, menuNo, menuName, menuPrice);
808         tempOrderList.remove(food);
```

```
818         modelOrdering.removeRow(row);
819         modelOrdering.fireTableDataChanged();
```

음식메뉴 데이터를 주문리스트에서 삭제하고, 테이블 모델 데이터도 삭제하고, 데이터 변경을

보여주기 위해 DefaultTableModel의 fireTableDataChanged() 메소드를 호출했습니다.

4. 관리자메뉴, 주문조회 매출내역 = 오건철, 최장원

주문조회는 로그인된 User의 가장 최신 주문내역을 보여주는 기능입니다. 주문할때마다 주문리스트 데이터가 덮어씌워 집니다. 주문내역 누적 및 날짜별 조회기능은 시간관계상 구현이 안됐는데, 시간 날때 개인적으로 구현해 보겠습니다.

주문조회 화면은 CardLayout의 카드패널에 만들었습니다. 네비게이션 bar의 메뉴 아이콘을 누르면 해당 카드로 이동합니다. Map<Food, Integer>orderList에 저장된 데이터를 출력합니다.

관리자 메뉴는 관리자로 로그인할때만 보이고, 클릭시 새로운 JFrame 창이 생성되어, 2장의 카드(cardSalesResult, cardManageMenu)를 가지고 있습니다.

```
217         List<Map.Entry<Integer, Food>> sortedSales = new ArrayList<Map.Entry<Integer, Food>>();
218
```

총매출(salesResult)는 top5메뉴와 같은 방법으로, sortedSales에 데이터를 넣고, List를 Collections.sort를 이용하여 정렬합니다.

메뉴관리 카드패널은(cardManageMenu) 전체 메뉴를 테이블로 보여주고, 추가 수정 삭제 버튼으로 데이터를 변경할수 있도록 했습니다.

메뉴 추가 혹은 변경시 새로운 JFrame을 만들어 보여줍니다.

```
716         int max = -1;
717         List<Integer> menuNos= new ArrayList<Integer>();
718         while(itr.hasNext()) {
719             food = itr.next();
720             if(food.getMenuCategory().equals(menuCategory)) {
721                 menuNos.add(food.getMenuNo());
722                 max = Math.max(food.getMenuNo(), max);
723             }
724         }
725
726         int newMenuNo = -1;
727         for(int i =max; i>0; i--) {
728             if(!menuNos.contains(i)) {
729                 newMenuNo = i;
730             }
731         }
732         if(newMenuNo == -1)
733             newMenuNo = max+1;
```

메뉴추가시에 menuNo 음식 번호는 1부터 max 메뉴번호사이에서 가장 작은 정수를 메뉴번호가 자동 지정되도록 구현했습니다.

```
770         tableModel.addRow(new String[] { food.getMenuCategory(),
771                         String.valueOf(food.getMenuNo()),
772                         food.getMenuName(),
773                         String.valueOf(food.getMenuPrice()),
774                     });
775         tableModel.fireTableDataChanged();
```

```
779         String[] newRow = new String[] { food.getMenuCategory(),
780                     String.valueOf(food.getMenuNo()),
781                     food.getMenuName(),
782                     food.toCurrency(food.getMenuPrice()) };
783         switch(food.getMenuCategory()) {
784             case "NOODLE": modelN.addRow(newRow); modelN.fireTableDataChanged(); break;
785             case "SOUP": modelS.addRow(newRow); modelS.fireTableDataChanged(); break;
786             case "RICE": modelR.addRow(newRow); modelR.fireTableDataChanged(); break;
787         }
```

음식메뉴 데이터 변경후, 테이블 모델 데이터 변경을 적용하기 위해 model.fireTableDataChanged()메소드를 호출했습니다. 메인화면의 테이블도 함께 업데이트 하여 보여주도록 하였습니다.

```
867         Food deleteFood = new Food(menuCategory, menuNo, oldMenuName, oldMenuPrice);
868         Food newFood = new Food(menuCategory, menuNo, newMenuName, newMenuPrice);
869
870         FoodMenu menu = userRepo.getFoodMenu();
871         menu.removeFood(deleteFood);
872         menu.addFood(newFood);
873
874         tableModel.removeRow(row);
875         tableModel.addRow(new String[] {
876                 menuCategory,
877                 String.valueOf(menuNo),
878                 newMenuName,
879                 String.valueOf(newMenuPrice),
880                     });
881
882         tableModel.fireTableDataChanged();
883
884         modifyFrame.dispatchEvent(new WindowEvent(modifyFrame, WindowEvent.WINDOW_CLOSING));
885
886         //Update Table : Add food
887         switch(menuCategory) {
888             case "NOODLE": removeRow(modelN, deleteRow); modelN.fireTableDataChanged(); break;
889             case "SOUP": removeRow(modelS, deleteRow); modelS.fireTableDataChanged(); break;
890             case "RICE": removeRow(modelR, deleteRow); modelR.fireTableDataChanged(); break;
```

메뉴 삭제 혹은 변경시에도 메인화면의 당 메뉴 테이블을 변경 하였습니다.