Timothy Liu

Dis 2A

005994409

While working on my program, one problem was allowing premier plan users to eat multiple times in a single meal period, while restricting regular plan users from doing so. I had to add a check for whether or not they were on premier, and if they were on a premier plan and they have already eaten during the meal period, then they can swipe in during the same meal period again, but if they were on regular then they couldn’t swipe in during the same meal period.

Another problem I encountered was trying to reset the number of meals each week for regular users and not allow their meals to carry over. I had to create a private data member called “int mMealsLeftThisWeek” This would keep track of how many meals regular plan users had throughout the week. When a new week occurred, the value of howManyMealsLeft would subtract the value of mMealsLeftThisWeek, which discards the unused meal swipes for regular users. For premier users, the value of mMealsLeftThisWeek was set equal to howManyMealsLeft so it did not affect premier plan users.

|  |  |  |
| --- | --- | --- |
| Test Case | Handles Correctly or Not | Comments |
| BruinCard noMealPlanBruinCard;  MealPlan noMealPlan;    assert( noMealPlanBruinCard.eat( BREAKFAST)== **false**);  assert( noMealPlanBruinCard.eat( LUNCH )== **false**);  assert( noMealPlanBruinCard.eat( DINNER )== **false**);  assert( noMealPlanBruinCard.eat( WEEKENDBRUNCH )== **false**);  noMealPlanBruinCard.startQuarter();  assert( noMealPlanBruinCard.eat( BREAKFAST)== **false**);  noMealPlanBruinCard.newWeek();  assert( noMealPlanBruinCard.eat( BREAKFAST)== **false**);  assert( noMealPlanBruinCard.eat( BREAKFAST)== **false**);  noMealPlanBruinCard.newDay();  assert( noMealPlanBruinCard.eat( BREAKFAST)== **false**); | Handles Correctly | Bruin Card without meal plan |
| BruinCard premierPlanUser;  MealPlan premierPlanUserMealPlan;  premierPlanUserMealPlan.pickPlan(PREMIER19);  premierPlanUser.purchaseMealPlan(premierPlanUserMealPlan);  premierPlanUser.startQuarter();  premierPlanUser.newWeek();  premierPlanUser.newWeek();  premierPlanUser.newWeek();  premierPlanUser.newWeek();  premierPlanUser.newWeek();  premierPlanUser.newWeek();  premierPlanUser.newWeek();  premierPlanUser.newWeek();  premierPlanUser.newWeek();  premierPlanUser.newWeek();  premierPlanUser.newWeek();    assert( premierPlanUser.mealsLeftThisWeek() == 11\*19);  assert( premierPlanUser.eat( BREAKFAST ) == **true**);  assert( premierPlanUser.eat( LUNCH ) == **true**);  assert( premierPlanUser.eat( BREAKFAST ) == **true**);  assert( premierPlanUser.mealsLeftThisWeek() == 11\*19-3); | Handles correctly | Not swiping in for 11 weeks on Premier Plan |
| BruinCard regularPlanUser;  MealPlan regularPlanUserMealPlan;  regularPlanUserMealPlan.pickPlan(REGULAR19);  regularPlanUser.purchaseMealPlan(regularPlanUserMealPlan);    regularPlanUser.startQuarter();  regularPlanUser.newWeek();  regularPlanUser.newWeek();  regularPlanUser.newWeek();  regularPlanUser.newWeek();  regularPlanUser.newWeek();  regularPlanUser.newWeek();  regularPlanUser.newWeek();  regularPlanUser.newWeek();  regularPlanUser.newWeek();  regularPlanUser.newWeek();  regularPlanUser.newWeek();      assert( regularPlanUser.mealsLeftThisWeek() == 0); | Handles Correctly | Not swiping in for 11 weeks on Regular Plan |
| BruinCard regular11B;  MealPlan regular11M;  regular11M.pickPlan(REGULAR11);  regular11B.purchaseMealPlan( regular11M );  regular11B.startQuarter();  assert (regular11M.cost()== 4646.64); | Handles correctly | Check cost of meal plan |
| BruinCard x;  MealPlan xM;  xM.pickPlan(REGULAR14);  x.purchaseMealPlan(xM);  x.startQuarter();  x.newWeek();  x.eat( BREAKFAST );  assert(x.mealsLeftThisWeek() == 13);  x.eat (LUNCH);  assert(x.mealsLeftThisWeek() == 12);  x.newWeek();  assert(x.mealsLeftThisWeek()==14);  x.eat( BREAKFAST);  x.eat(LUNCH);  x.eat(DINNER);  x.newDay();  x.eat(BREAKFAST);  assert(x.mealsLeftThisWeek() == 10); | Handles Correctly | Regular Plan user’s meals left are decremented with each meal swiped into |