GRSManager: Order Processing Automation

Business Problem

Summarize the problem this program is attempting to resolve by describing the specific challenges that exist and how the software solution addresses them.

* Why do research customers not email their order forms to a general email inbox like clinical customers?
  + The pricing for research customers is different from the standard commercial pricing offered to clinical customers and is different between research customers.
  + ERT is over 50% of the business and chooses to upload spreadsheets to a portal that Circassia employees can access to get new orders.
* What are some unique requirements of the GRS order process?
  + Only domestic shipments are entered into NAV. If an order is being shipped OUS, the information is copied over to a word document and emailed to Circassia Sweden order processing. Circassia Sweden enters the order in NAV.
  + In the case of domestic shipments, the order is entered into NAV and then an email with order and customer number is sent to Morrisville logistics informing them of a new order that needs packaging and scheduling.
  + There are 3-5 pre-entry (in NAV) steps that are required for GRS orders which require manually copy/pasting, re-entering, and reformatting order information from one program to another before being entered in NAV.

Analysis and Requirements

**Model Details**

The model organizes different elements of the order process into classes which are used to manage the creation, tracking, and storage of order data. It provides the customer and the vendor simple interfaces for entering and processing orders, respectively. Table 1 lists the non-GUI, non-utility classes used to represent different processes within order management. The default values for all the factors are listed in the Default Value column.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Symbol** | **Factor** | **Default Value** | **Range** | **Description** |
| x | Prey count | 10 (start) | 0 to 300 | Number of prey |
| a | Prey birth rate | 0.165 | 0.0 to 1.0 | Birth rate of prey |
| b | Prey death rate | 0.0006 | 0.0 to 1.0 | Death rate of prey caused by predation |
| y | Predator count | 250 (start) | 0 to 500 | Number of predators |
| c | Predator death rate | 0.23 | 0.0 to 1.0 | Death rate of predators |
| p | Predator birth rate | 0.00068 | 0.0 to 1.0 | Population growth rate of predator due to availability of prey |
| z | Scavenger count | 400 (start) | 4 to 500 | Number of scavengers |
| e | Scavenger death rate | 0.1 | 0.0 to 1.0 | Death rate of scavengers |
| f | Scavenger birth rate from kills | 0.000002 | 0.0 to 1.0 | Population growth rate from scavenging corpses of prey that die from predation |
| g | Scavenger birth rate from prey | 0.0003 | 0.0 to 1.0 | Population growth rate from scavenging corpses of prey that die naturally |
| h | Scavenger birth rate from predators | 0.0006 | 0.0 to 1.0 | Population growth rate from scavenging corpses of predators |
| i | Scavenger population limiting factor | 0.001 | 0.0 to 1.0 | Limits population growth of scavengers. *This value is fixed and cannot be changed.* |
| **Table 1. Default Simulation Factors/Parameters** | | | | |

**User Interface**

The user interface has text fields for input of all of values listed in Table 1 except for the last one (Scavenger population limiting factor). The user is expected to enter integer values for the counts and double values for the remaining rate factors. Actions are initiated via three buttons at the top of the application window.

When the user starts the simulation, the graph shows the progress of the simulation step-by-step, with three points plotted for each step: one for the predator count, one for the prey count, and one for the scavenger count. There are 600 steps. The points are color coded as illustrated by Figure 1, which uses red for predator counts, black for prey counts, and blue for scavenger counts.

When a simulation run is complete, the text fields for the counts show the final population count values. For example, in the snapshot above, the population counts began with their default values: 10, 250, and 400. Figure 1 shows the final counts: 244 predators, 343 prey, and 311 scavengers.

Requirements Use Cases

**Use Case 1: Startup**

***Preconditions:***The user has just started the program.

***Main Flow:***The login panel opens with the two blank text fields for user id and password.

The user enters their login credentials [S1]. The user clicks Submit [S2] to close the window. Figure 2 shows the the initial window with default values. Figure 3 shows the initial window after the user changes all species' names from their default values and the scavenger's color to green.

***Sub flows:***

* [S1] The user enters new species names (as desired) [S2].
* [S2] The user picks different colors (as desired) [S3].
* [S3] The user clicks Close. The main simulation window opens with an empty graph area and default simulation parameters in the text fields [UC2]. See Table 1 and Figure 4.

***Alternative Flows:***

* [E1] There is a non-number in a Count field when the user clicks Start (Figure 5). The program displays a dialog box warning of the error and the simulation does not run. [E5]
* [E2] There is a negative number in a Count field when the user clicks Start (Figure 6). The program displays a dialog box warning of the error and the simulation does not run. [E5]

When GRSManagerGUI is constructed it initializes the following in this order:

1. LoginPanel is constructed with GridBagLayout.
2. AdministratorPanel is constructed and initializes the following panels in this order:
   1. VendorEmployeeDirectoryPanel is constructed and initializes first instance of GRSManager by calling the static GRSManager method getInstance(). GRSManager contains a static instance variable of itself so the getInstance() method checks this instance variable for the current GRSManager and returns it if it exists. If it is null, no GRSManager has been constructed, it calls GRSManager’s private constructor via the *new* keyword.
      1. GRSManager constructs the following data structure classes in this order:
         1. NioxCatalog
            1. LinkedListRecursive<Product> - an empty product list
         2. CompanyDirectory
            1. LinkedListRecursive<Company> - an empty Company list
         3. OrderRecord
            1. String [][] – an empty 2D string array
         4. Administrator
         5. UserDirectory
            1. LinkedAbstractList<User> - an empty list of User
      2. After GRSManager’s construction and subsequent creation of the above objects, **they are now available for access by all other GUI components via the static getInstance() method. User actions call GRSManager methods, which may call data structure class methods.** VendorEmployeeDirectoryPanel initializes its own instances of UserDirectory and CompanyDirectory to GRSManager’s (which are empty lists of Users and Companies, respectively). VendorEmployeeDirectoryPanel sets up the rest of the GUI components for its panel and returns control to AdministratorPanel’s constructor.
   2. ResearchEmployeeDirectoryPanel
      1. Requests instance of GRSManager (previously constructed during VendorEmployeeDirectoryPanel’s request for UserDirectory and CompanyDirectory), and receives empty LinkedLists of users and companies.
      2. Sets up the rest of its GUI components and returns control to AdministratorPanel’s constructor.
   3. ProductCatalogPanel
      1. Requests same instance of GRSManager and gets the product catalog class (NioxCatalog) and sets its one instance of NioxCatalog to the empty product list that is returned.
      2. Constructs remaining GUI components and returns control to AdministratorPanel’s constructor.
   4. OrderRecordPanel
      1. Requests same instance of GRSManager and gets the empty list of users and companies by setting its own instance variables for UserDirectory and CompanyDirectory to those returned by GRSManager.
      2. Sets up GUI components and calls the updateTables() method. updateTables() calls the updateData() method in the private class OrderDirectoryTableModel.
      3. updateData() in OrderDirectoryTableModel requests GRSManager’s OrderRecord first and then a 2D array of data by calling getShortOrderInfo. At this point, GRSManager’s instance of OrderRecord has an empty 2D array of Strings so null is copied over into the 2D array returned by getShortOrderInfo.
      4. Control returns to AdministratorPanel which finishes setting up and adding GUI components. Control returns to GRSManagerGUI.
3. VendorPanel. The Login and Administrator Panels have been constructed. Along with them the panels VendorEmployeeDirectoryPanel, ResearchEmployeeDirectoryPanel, ProductCatalogPanel, and OrderRecordPanel are all added to a card layout in the AdministratorPanel. The GRSManager has been initialized with references to data structure classes.
   1. VendorPanel’s constructor initializes
      1. VendorCompanyOrderSchedulePanel which requests GRSManager’s currentUser instance variable. At this point it is null because no one has logged into the system. It also requests the NioxCatalog from GRSManager which has an empty LinkedList of products.
      2. Sets up GUI components and calls the updateTables() method which calls the udpateData() methods for
4. ResearchPanel

The VendorCompanyOrderSchedulePanel is the interface where a vendor can view, select, and mark complete a list of open orders. It consists of a JPanel with GridBagLayout and one button in the far left corner labeled “Processed.” Below this JPanel is a JTable with order information for orders that are open. An order is open if it has not been processed by the vendor. This list of open orders is the vendor’s order schedule. When an order is processed, the vendor selects that order in the table and clicks the “Processed” button. Internally, the Order’s status is changed to “Processed,” and the Order is removed from the table.

**Use Case 18: Vendor Interacts with an Open Order Schedule**

***Preconditions:*** An administrator has previously setup a vendor employee and a research company employee as users of the system. The administrator has loaded product information and order records. The research company employee has logged in and entered at least one order. A vendor employee is currently logged in.

***Main Flow:*** The user selects the open order [S1] and clicks the “Processed” button[S2].

***Sub Flows:***

* [S1] The user selects the open order displayed in the order schedule
* [S2] With the order selected, the user clicks the “Processed” button

***Alternative Flows:***

* [E1] There is no order selected in the order schedule and the user clicks the “Processed” button. The program displays a dialog box warning with the text “No order has selected.”