CSC216: Project 2

Project 2, Part 1: PackDoption

Project 2, Part 1: PackDoption

Version with no table of contents (for printing). (project2-part1-print)

Project 2 requires you to go through standard software development phases to design, implement, and test a complete Java program consisting of multiple source code files and JUnit test cases.

Deliverables and Deadlines

The project comes in two parts. Deliverables for Part 1 are:

1. Design document with incorporated UML class diagram
2. Black box test plan document

Part 1 Due Date: Wednesday, July 3, 2024 at 11:45 PM

Late Deadline (Design): Friday, July 5, 2024 at 9:00 AM

Late Deadline (BBTP): Saturday, July 6, 2024 at 11:45 PM

Reminder

You will *not* be working with a partner for Part 1 of *any* project. All work must be strictly your own.

Problem overview

Several animal rescue groups (listed as rescues in the system) have hired you to develop software to track all of the animals (dogs and cats) currently available for adoption along with those that have been adopted through the rescue groups. Animal rescue groups track adopted and available animals, and they remain in contact with owners after adoption for special events (such as reunions). Each rescue group works with a veterinarian, who sees animals for appointments. The system contains a queue of appointments.

The application that you must create has a graphical user interface (GUI) as shown in Figure 1. The GUI contains two different views: (1) rescues view [Figure 1(b)] and (2) animal view [Figure 1(c)].

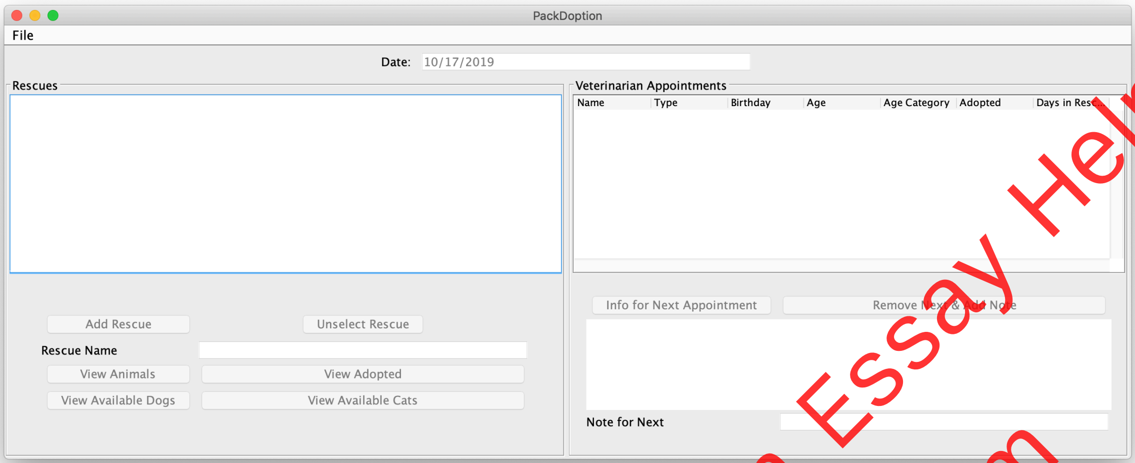
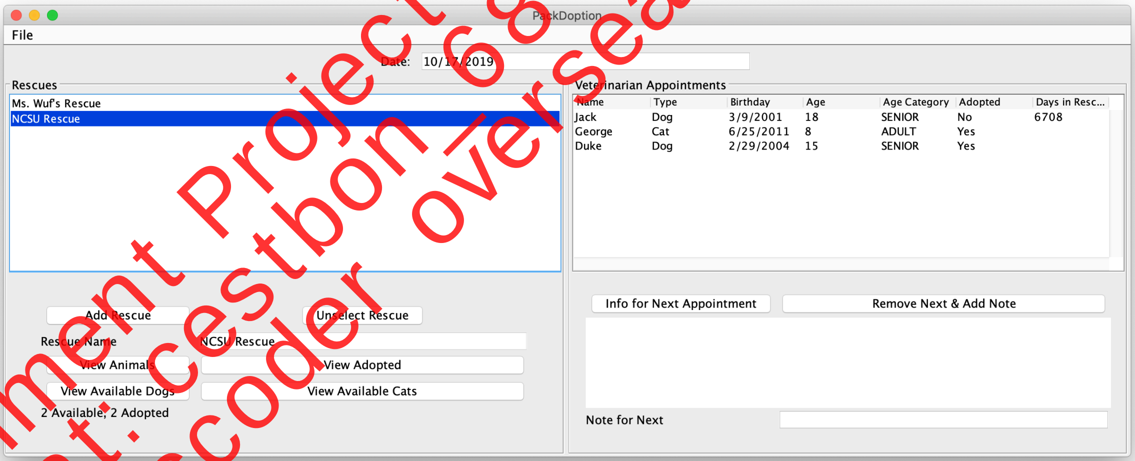


Figure 1(a): PackDoption GUI displaying empty rescue list.



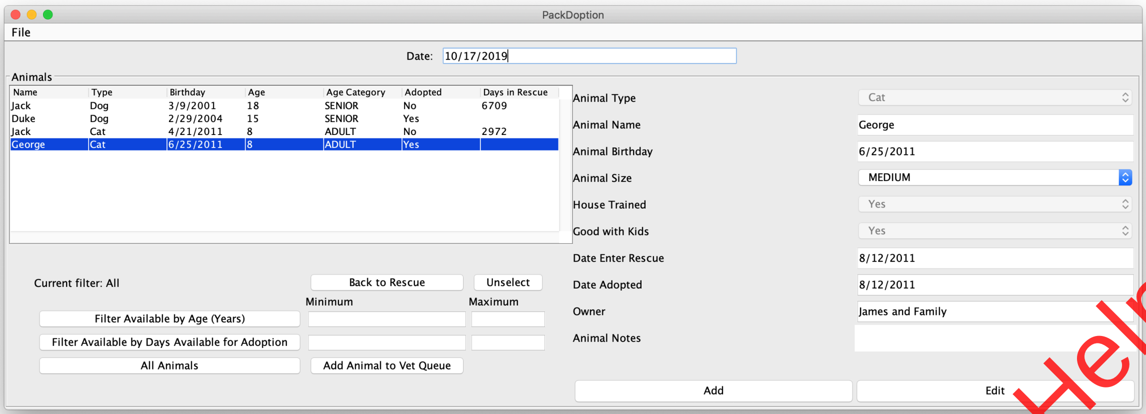
Ms. Wuf's Rescue
NCSU Rescue

Name	Type	Birthday	Age	Age Category	Adopted	Days in Resc...
Jack	Dog	3/9/2001	18	SENIOR	No	6708
George	Cat	6/25/2011	8	ADULT	Yes	
Duke	Dog	2/29/2004	15	SENIOR	Yes	

Rescue Name: NCSU Rescue

2 Available, 2 Adopted

Figure 1(b): PackDoption GUI displaying two rescues in addition to three veterinarian appointments for the selected rescue, NCSU Rescue.


PackDoption
File
Back to Rescue
Unselect
Filter Available by Age (Years)
Filter Available by Days Available for Adoption
All Animals
Add Animal to Vet Queue
Add
Edit

Date: 10/17/2019

Animals

Name	Type	Birthday	Age	Age Category	Adopted	Days in Rescue
Jack	Dog	3/9/2001	18	SENIOR	No	6709
Duke	Dog	2/29/2004	15	SENIOR	Yes	
Jack	Cat	4/21/2011	8	ADULT	No	2972
George	Cat	6/25/2011	8	ADULT	Yes	

Current filter: All

Minimum Maximum

Animal Type	Cat
Animal Name	George
Animal Birthday	6/25/2011
Animal Size	MEDIUM
House Trained	Yes
Good with Kids	Yes
Date Enter Rescue	8/12/2011
Date Adopted	8/12/2011
Owner	James and Family
Animal Notes	

Figure 1(c): PackDoption GUI with animals.

Requirements

The system will track different animal rescue groups (rescues) with their associated animals and veterinarian appointments.

Each rescue should include: a rescue name, a list of animals, and queue of vet appointments. Once a rescue is created in the system, the rescue name cannot be edited.

Each animal (dog or cat) should include: name, birthday, size, whether the animal is house trained, whether the animal is good with kids, list of notes about the animal, when the animal entered the rescue, and whether the animal has been adopted. If the animal has been adopted, the system stores adoption date and owner information. Once an animal has been added to the system, the user can add notes and update animal size and adoption information. Each view of the system has a date field listed, which is initially set to today's date; the date field must be after all other dates in the system (e.g., birthday, date enter rescue, adoption date). This field is used for calculating animal ages and days animal has been available for adoption. Based on age and size, the age category (young, adult, senior) is calculated. Cats and small dogs are considered adult at 4 years old and senior at 9 years old. Medium dogs are considered adult at 3 years old and senior at 9 years old. Large dogs are considered adult at 3 years old and senior at 6 years old.

Dates should be stored as a month, a day, and a year (M/D/YYYY). Each note should have a date and a text note.

Veterinarian appointments are stored as a queue of animals without appointment times.

The following use cases describe the interactions between the PackDoption system and its users. For this system, there is only one actual user, the person who manages the data. [Later iterations of the project, which are outside of the scope of CSC216, will allow multiple users.]

- Use Case 1: Startup and Shutdown
- Use Case 2: Open a PackDoption data file
- Use Case 3: Save a PackDoption data file
- Use Case 4: Add rescue to system

- Use Case 5: View animals
- Use Case 6: Filter animals
- Use Case 7: Manipulate animal information
- Use Case 8: Add animal to vet queue
- Use Case 9: Remove animal from vet queue

Use Case 1: Startup and Shutdown

Preconditions: None

Main Flow: The user starts the program [S1][S2]. The user can (a) start a new PackDoption data file (containing rescues with associated animals and veterinarian appointments), (b) open an existing PackDoption data file [UC2], (c) save current rescues and associated data to a PackDoption data file [UC3], (d) add rescue to system [UC4], (e) view animals in rescue [UC5], (f) filter animal information [UC6], (g) add and/or edit animal information [UC7], (h) add animals to veterinarian appointment queue [UC8], and (i) remove animals from veterinarian appointment queue [UC9]. When the user is done, the user either closes the window or selects *File-> Exit* [S3][S4]. The program checks to be sure the current data has been saved [S5], then quits execution [S6].

Subflows:

- [S1] The program starts, displaying the PackDoption user interface as shown in Figure 1(a). All buttons and text fields are disabled. Date field is initially set to today's date.
- [S2] All buttons and text fields are disabled until the user (1) starts a new PackDoption data file *File-> New* or (2) opens an existing PackDoption data file *File-> Load* [UC2].
- [S3] The user closes the window to exit the program [E1].
- [S4] The user clicks *File-> Exit* to exit the program [E1].
- [S5] If the current data file has changed since it was opened or last saved, the user is prompted to save the file [UC3].
- [S6] The program stops execution with no errors or exceptions.

Alternative Flows:

- [E1] If the user aborted the file save operation, the program does not exit [UC2].

Use Case 2: Open a PackDoption data file

Preconditions: The program is running [UC1], and the user interface is open as shown in Figure 1.

Main Flow: The user selects *File->Load* from menu bar [S1]. The user then browses for and selects the desired data file from a File Open dialog [S2]. The file is loaded [S3] and the rescues are listed.

Subflows:

- [S1] A File Open dialog is displayed (Figure 2), and the user browses for and selects the desired data file.
- [S2] The data file is opened and read [S3], then the system populates the user interface (Figure 1(b) and Figure 1(c)) [E1].

- [S3] A valid PackDoption data file has rescue information, animal information, and veterinarian appointment queue. A valid PackDoption data file is structured as shown below and has extension *.md* (see sample document (./assets/part1/test-files/sample.md)). The first animal line below shows an animal who has been adopted so has fields for adoption information; whereas the second animal line shows an animal currently available for adoption.

```
# Rescue names
* Dog/Cat,Name,Birthday,Size,House-Trained,Good With Kids,Date enter shelter,Adopted-true,
* Dog/Cat,Name,Birthday,Size,House-Trained,Good With Kids,Date enter shelter,Breed (if dog
- name, birthday
```

Alternative Flows:

- [E1] If the selected file is not a valid PackDoption data file or an error occurs while reading the selected file, an error dialog opens with message "Error opening file." When the user clicks *OK*, the dialog closes, and the program returns to the previous display.

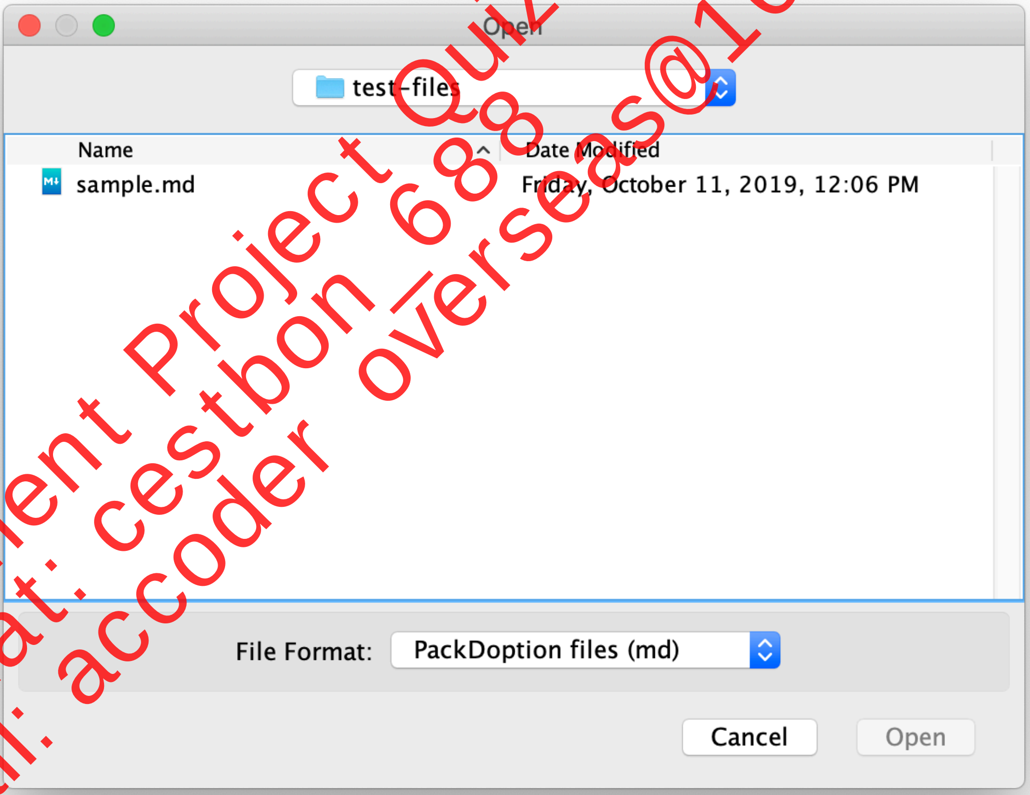


Figure 2: File Open dialog.

Use Case 3: Save a PackDoption data file

Preconditions: The program is running. The user has started a new PackDoption data file or opened an existing one [UC1, S2].

Main Flow: The user selects *File -> Save* [S1][S2]. The user clicks *Save* to save the data file [E3][E4].

Subflows:

- [S1] The user selects *File -> Save*, and a File Save dialog opens to the path and filename (a) used to open the file or (b) used at the last file save operation [E1][E2]. The file system browser dialog is opened with a filter on the default file extension (**.md*).
- [S2] Rescues are sorted by rescue name. Each rescue starts with a `#` followed by a space and then the rescue name. Animals are sorted by birthday then name. Each animal is on its own line with a leading `*`, a space, animal information in comma separated list. Each veterinarian appointment is on its own lines with a leading `-`, a space, the animal's name, a comma, and the animal's birthday; each appointment must be associated with an animal in the animal list. A blank line follows before the next rescue is listed. (See example file in [UC2][S3])

Alternative Flows:

- [E1] If the user clicks *Cancel* on the File Save dialog, the dialog closes, aborting the file save operation [UC2].
- [E2] If the file has not been saved before, the File Save dialog opens to the default data file storage location (i.e., `./`) with the filename used to load or create new data file [S1].
- [E3] If the filename is blank, just whitespace, or does not end with .md when *Save* is clicked, an error dialog is displayed to the user with the message "File not saved" [E5].
- [E4] If an error occurs while writing the data file to disk, and error dialog is displayed [E5].
- [E5] The user clicks *OK*, closing the dialog and aborting the file save operation [UC2].

Use Case 4: Add rescue to system

Preconditions: The program is running. The user has started a new PackDoption data file or opened an existing one [UC1, S2].

Main Flow: The user can add [S1] a rescue.

Subflows:

- [S1] When no rescue is selected (click the *Unselect Rescue* button if necessary), the user fills in the rescue name text field. The user then clicks *Add Rescue* button. The rescue is then added to the sorted rescues list with no rescues selected. The rescue detail area is cleared.
- [S2] When a rescue is selected, the veterinarian appointment queue is displayed. The queue contains all upcoming appointments.

Alternative Flows:

- [E1] If the user clicks *Add Rescue* with an empty (or containing only whitespace) name, an error dialog is shown with message "Cannot add Rescue with empty name". The user clicks *OK* on the error dialog.

Use Case 5: View animals

Preconditions: The program is running. The user has started a new PackDoption data file or opened an existing one [UC1, S2].

Main Flow: The user can view the animals for a selected rescue in the system.

Subflows:

- [S1] The user selects a rescue.
- [S2] If the user clicks a *View Animals* button, all animals for the rescue are shown in the animal table (Figure 1(c)).
- [S3] If the user clicks *View Adopted*, all animals adopted from the selected rescue are shown in the animal table.
- [S4] If the user clicks *View Available Dogs*, all dogs currently available for adoption from the selected rescue are shown in the animal table.
- [S5] If the user clicks *View Available Cats*, all cats currently available for adoption from the selected rescue are shown in the animal table.

Use Case 6: Filter animals

Preconditions: The program is running. The user has started a new PackDoption data file or opened an existing one [UC1, S2]. The user is viewing animals for a selected rescue [UC5].

Main Flow: The user can filter the animals for a selected rescue in the system.

Subflows:

- [S1] The user enters filter criteria: (1) minimum and maximum age in years [E1] [E2] OR (2) minimum and maximum days available for adoption [E3] [E4].
- [S2] The user clicks the *Filter* button: (1) *Filter Available by Age (Years)* OR (2) *Filter Available by Days Available for Adoption* . All of the animals for the selected rescue that meet the filter criteria are displayed in the animal table.
- [S3] If the user clicks *Back to Rescue*, the user interface returns to the rescue list.

Alternative Flows:

- [E1] If the user clicks *Filter Available by Age (Years)* with a non-integer minimum or maximum age, an error dialog is shown with the message "Age ranges are not integers." The user clicks *OK* and is returned to the user interface with all animals for the rescue displayed.
- [E2] If the user clicks *Filter Available by Age (Years)* with maximum age is less than minimum age or ages are negative, an error dialog is shown with the message "Age ranges are not valid." The user clicks *OK* and is returned to the user interface with all animals for the rescue displayed.
- [E3] If the user clicks *Filter Available by Days Available for Adoption* with a non-integer minimum or maximum days, an error dialog is shown with the message "Day ranges are not integers." The user clicks *OK* and is returned to the user interface with all animals for the rescue displayed.
- [E4] If the user clicks *Filter Available by Days Available for Adoption* with maximum day is less than minimum day or days are negative, an error dialog is shown with the message "Day

ranges are not valid.." The user clicks *OK* and is returned to the user interface with all animals for the rescue displayed.

Use Case 7: Manipulate animal information

Preconditions: The program is running. The user has started a new PackDoption data file or opened an existing one [UC1, S2]. The user is viewing animals for a selected rescue [UC5].

Main Flow: The user can add [S1] and update (only certain fields) [S2] animal information.

Subflows:

- [S1] When no animal is selected (click the *Unselect* button if necessary), the user fills in the animal detail text fields. The user then clicks *Add* button. The animal is then added to the displayed animal list with no animals selected. The animal detail area is cleared.
- [S2] The user selects the animal to be edited. The animal is highlighted (table in top left portion of animal view in Figure 1(c)), and the animal details are displayed in the animal edit area (right portion of animal view in Figure 1(c)). The user edits the data (animal size and adoption information only) then clicks *Edit* to update the animal in the animal list [E1].

Alternative Flows:

- [E1] If the user clicks *Add* or *Edit* with any invalid values, an error dialog is shown with message "Cannot add: <error message related to invalid value- from exception>." The user clicks *OK* on the error dialog.

Use Case 8: Add animal to vet queue

Preconditions: The program is running. The user has started a new PackDoption data file or opened an existing one [UC1, S2]. The user is viewing animals for a selected rescue [UC5].

Main Flow: The user can add an animal to the veterinarian queue.

Subflows:

- [S1] The user selects an animal to add to the vet queue.
- [S2] The user clicks *Add Animal to Vet Queue*.

Use Case 9: Remove animal from vet queue

Preconditions: The program is running. The user has started a new PackDoption data file or opened an existing one [UC1, S2].

Main Flow: The user can (1) view animal information for next appointment and (2) remove next appointment and add note.

Subflows:

- [S1] The user clicks *Info for Next Appointment*. Information for the next appointment is displayed (animal's name, animal's birthday, and animal's notes).
- [S2] The user fills in the note for next appointment and clicks *Remove Next & Add Note*.

- [S3] First animal (next) is removed from queue and notes is added to animal with date field from user interface.

Alternative Flows:

- [E1] If there are no appointments in the queue, *Info for Next Appointment* and *Remove Next & Add Note* buttons are disabled.
- [E2] If the user clicks *Remove Next & Add Note* with a whitespace only note, an error dialog is shown with message that includes "Cannot remove appointment". The user clicks *OK* on the error dialog.

Design

Trying to jump from requirements right into coding without doing some design first is bound to lead to trouble. You will eventually implement our teaching staff design (Project 2 Part 2), but first you need to propose your own. To do this, we give you a scenario and insist on some restrictions for your design.

Scenario

Now that you have the requirements you can propose a design to create an application that will help the animal rescue group keep track of animals!

Assignment

You must design an application that satisfies the requirements of the PackDoption application. You will create a design proposal that shows the objects, their states and behaviors, and the relationships among the objects needed to implement the requirements. Your design must be described in document containing a design rationale and a UML class diagram.

Your design should:

- utilize the Model-View-Controller (MVC) design pattern (see the note about MVC, below),
- utilize one additional design pattern,
- contain at least one interface or abstract class,
- contain at least one inheritance relationship,
- contain at least one composition relationship,
- contain one *custom* ArrayList, and
- contain one *custom* LinkedList.

To help your evaluate your design, you should answer the following technical questions in your design document as part of the rationale:

1. What objects are important to the system's implementation and how do you know they are important?
2. What data are required to implement the system and how do you know these data are needed?
3. Are the responsibilities assigned to an object appropriate for that object's abstraction and why?

4. What are the relationships between objects (such as inheritance and composition) and why are those relationships are important?
5. Have you identified any design patterns appropriate for implementing the system (i.e., the State Pattern)? How are the identified patterns appropriate for implementing the system?
6. What are the limitations of your design? What are the constraints of your system?

MVC Note

Java Swing, the user interface (UI) libraries for Java, does not follow the strictly traditional definition of MVC. Instead, Java Swing utilizes what might be called a *separable model architecture (http://www.oracle.com/technetwork/java/architecture-142923.html)*. This means that the model is separate and distinct from the view/controller classes that make up the UI. For your design, you will focus on the Model. In your UML class diagram, you should represent the UI as a class with no state or behavior. Your diagram should also show which class(es) your UI will interact with through some type of composition/aggregation/association relationship. The relationship between your model and the UI ***must*** be justified in your design rationale.

When thinking about the relationships between your UI and the model, consider the following questions:

1. What are the data and behaviors of your model that will be shown through the UI?
2. How does your UI get those data to display; what methods of the model must be called?

Format

Submit your design proposal via Gradescope (https://gradescope.com/) under the assignment named **P2P1: Design**.

Use the provided design proposal template (../assets/DesignProposal_Template.docx) as a starting point for your design proposal. Use a UML diagramming tool, (options are listed the Software Development Practices notes (https://pages.github.ncsu.edu/engr-csc216-staff/CSC216-SE-Materials/se-overview/#design-practices)), to create your UML diagram. Incorporate your UML diagram into your written proposal (using an editing tool such as MS Word) and save the entire document as a PDF. Alternatively, create a PDF for the design proposal and another PDF for the UML diagram, then append the diagram to the proposal to make a single PDF document.

Need a little more direction?

See the following example design proposal: Sample Design Proposal (../assets/SampleDesignProposal1.pdf).

Testing

This project requires you to do white box and black box testing. You can defer white box testing until Part 2 of this project. But now, you need to prepare some black box test cases. We will start you off with a scenario.

Scenario

Your manager has requested a black box test plan that describes five test cases that will determine if the finished program is ready to ship to the customer. Your manager wants you to write the tests before you begin development to clarify the inputs to and outputs from the system. Each test must demonstrate that the program satisfies a scenario from the requirements. A scenario is one major path of functionality as described by the requirements.

Assignment

You will write at least five (5) tests for the PackDoption project. Use the provided black box test plan template (../assets/STP_Template.docx) to describe your tests. Each test must be repeatable and specific; all input and expected results values must be concrete. All inputs required to complete the test must be specified. Additionally, you must provide instructions for how a tester would set up, start, and run the application for testing (What class from your design contains the main method that starts your program? What are the command line arguments, if any? What do the input files contain?). Describe the instructions at a level where anyone using Eclipse could run your tests.

Sample test input file (assets/part1/test-files/sample.md)

Format

You must submit your black box test plan via your section's submission platform as a PDF. Use the provided template (../assets/STP_Template.docx) as a starting point for your black box test plan. Save your BBTP as a PDF. Submit the PDF to Gradescope (https://gradescope.com/) under the assignment named **P2P1: BBTP**.

Need a little more direction?

See the following example black box test plan: Sample Black Box Test Plan (../assets/SampleBBTP1.pdf).

Deployment

For this class, deployment means submitting your work for grading. For Part 1 of this programming assignment, you must submit two PDF documents:

1. Design document with incorporated UML class diagram.
2. Black box test plan document.

Make sure that your submissions satisfy the grading rubrics (../assets/gradesheet.html).

You should submit the documents for Project 2 Part 1 to Gradescope (https://gradescope.com/).

Submission Reminders

The electronic submission deadline is precise. Do not be late. You should count on last minute system failures (your failures, ISP failures, or Gradescope failures). You are able to make multiple submissions of the same file. (Later submissions to a Gradescope assignment in overwrite the earlier ones.) To be on the safe side, start submitting your work as soon as you have completed a substantial portion.