W23 Final Project

Department of Electrical and Software Engineering
Schulich School of Engineering

The purpose of this final project is to demonstrate your understanding of software development principles using the Java programming language. This assignment is worth 30% of your final grade.

Final Project – Learning Outcomes

- Design and document an object-oriented Java application
- Connect to and import data from a given database
- Process and output data according to user input
- Incorporate GUI interaction with users

At the conclusion of the project, you will have a documented, modelled, and tested application that can be incorporated into an interview portfolio.

Project Logistics

You must work in a team of 3 or 4 members. Team formation guidelines are posted to D2L. You may not switch teams after the project has begun. You must use a **private** GitHub repository to collaborate and record your contributions[1]. Public repositories will be considered a violation of academic integrity. You will need to provide your repository URL and access to the instructors. More information is given in the deliverable descriptions.

Each team is assigned their own D2L dropbox. When submitting, ensure that all names are listed in the description. Only one group member needs to submit, but it is recommended that you complete the submission process together to double-check that nothing is missed! **If the submitting group member uploads the wrong files, it will impact the grade of all members.**

Project Deliverables and Timeline

The project deliverables are designed to give you iterative feedback throughout the development process. Each component contributes to the overall project grade. All deliverables are team-based.

You may optionally choose not to deliver one, or both, of the early deliverables (UML draft, test design). If a deliverable is not submitted, the weight will be added to the final package (e.g., if the UML draft is not submitted but the test design is, the final package will be worth 90% of the project grade). If a deliverable is submitted, the weight will not be shifted to the final package regardless of the grade that the deliverable receives.

The two initial deliverables are your opportunity to receive feedback on your project design and it is highly recommended that you complete them. We will not be able to give you detailed feedback except through the scheduled deliverables, and no extensions will be granted/no late submissions will be

[1] If students have privacy concerns about using GitHub to host their work, they can contact the course instructors. We can accommodate a self-hosted (by the team) repo as an alternative, but this must be discussed in advance.

accepted. The scoring for the initial deliverables will be more lenient than the scoring for the final project, and sticking with the due dates can keep the project on track.

Deliverable	**ENSF 380 Due Date**	**ENSF 409 Due Date**
Early UML Draft (5%)	March 9th at 5:00 PM	March 10th at 4:00 PM
Test Design (10%)	March 23th at 5:00 PM	March 24th at 4:00 PM
Final Package (85%) • UML Diagram (15%) • Code Implementation (40%) • Documentation (10%) • Unit Tests (25%) • Video Demonstration (10%)	April 6th at 5:00 PM	April 6th at 5:00 PM Dropbox closes April 7th at 4:00 PM; no penalty for submissions made during grace period

Additional Expectations

- You provide your **private** project GitHub URL at the time of the first deliverable (submit as a comment to the dropbox or as a text file, even if you are not submitting the deliverable).
- When you submit your final package, you invite the course instructor(s) to contribute. Please do not send an invitation before the final submission as these will expire; we will look at the repository if there are concerns about the distribution of work.

Client Problem Definition

Wildlife rescue centres provide care and rehabilitation for injured and orphaned wild animals. The tasks associated with caring for the animals depends on the types of animals in residence (e.g., feeding times), as well as the individual animal's specific care needs (e.g., an open wound requires daily re-bandaging). Your client, Example Wildlife Rescue (EWR), needs a program which can be used to generate a daily list of tasks for volunteers to complete in order to support the animals in residence. EWR specializes in medium-sized animals and refers other cases to other centres. They have facilities to support coyotes, foxes, porcupines, beavers and raccoons[2].

Sara, the volunteer scheduler from EWR explained: "Some tasks, like administering antibiotics, need to be done at the same time each day for maximum effectiveness. We go with whatever the vet specifies when it comes to medical care.

"With other tasks, like feeding, there's more flexibility, and with cage cleaning we do it once a day whenever there's room in the schedule. I try to come up with a schedule that is efficient, but the priority is always about ensuring the animals are cared for. I also try to calling in the backup volunteer because

[2] Note that information provided in this document about wildlife rescue centres, medical treatment, and animal habits may not be accurate. The project is designed to showcase object-oriented programming, not to serve as a reference for animal care.

we don't want to see burnout. I'd rather have the schedule be a little less efficient than call in the backup when it isn't absolutely essencial for getting the work done in time."

EWR's current manual process involves Sara drawing up a schedule each night for the next day. The schedule is specified at the granularity of 1 hour, with a list of tasks which should be completed within that hour. First she assigns all tasks with no flexibility, then the remaining tasks are scheduled in the remaining time. Where it is more efficient to perform similar tasks together, such as feeding all the coyotes at once, Sara will try to group these tasks. Because volunteer shifts begin and end on the hour, a task which spans two hours needs to be split into two tasks, with the associated preparation tasks being scheduled for each hour.

Currently, creating the schedule is time-consuming and prone to error, which is why EWR wants to automate it.

Usually only one volunteer is scheduled for each hour, but a backup volunteer can be called in if it is impossible to fit all the tasks in otherwise (e.g., if there are 2 medical treatments that must be performed at 11 AM, and each takes 35 minutes, they will both be scheduled for 11 AM, with a note to call the backup volunteer in). In such cases where the backup volunteer must be called in anyway, the rest of the hour can be filled with the assumption that two people will be available:

Time	Task	Time spent	Time available
11 AM *	Medical treatments	70	50
* Backup volunteer required			

Sara gave the following simplified example of her manual process for the hours between 7 and 9 PM:

- Feeding the coyotes takes 10 minutes of prep time, and 5 minutes to feed each coyote. If there are 8 coyotes and they are fed together, it takes 10 + 5*8 minutes (50 minutes). However, if 4 are fed at one time, and the remaining four are fed later, it takes 10 + 5*4 + 10 + 5*4 minutes (60 minutes). We usually try to feed them within a 3-hour window, between 7 and 9 PM.
- There is currently 1 fox with a leg wound which requires rebandaging every evening at 7 PM. This takes 20 minutes.
- There are orphaned kits which require bottle feeding every 2 hours. It takes 30 minutes to feed them all, and they need to be fed at the same time.
- The porcupines take 5 minutes each to feed, and there are currently 5 of them. They are fed between 7 and 9 PM.

Sara would first schedule the bandage change for 7 PM, because it has the least flexibility.

Time	Task	Qty	Time spent	Time available
7 PM	Rebandage fox leg wound	-	20	40

Feeding the coyotes together is most efficient, but because there isn't enough time left at 7 PM, she would schedule them for 8 or 9 PM, after considering other tasks with less flexibility. Because the kits have a smaller window for feeding, and there is sufficient time for them at 7 PM, Sara would schedule them.

Time	Task	Qty	Time spent	Time available

7 PM	Rebandage fox leg wound	-	20	40
	Feed kits	-	30	10

With only 10 minutes remaining at 7 PM, Sara would schedule feeding 2 of the porcupines.

Time	Task	Qty	Time spent	Time available
7 PM	Rebandage fox leg wound	-	20	40
	Feed kits	-	30	10
	Feed porcupines	2	10	0

Sara would then look at what tasks remained, and repeat the process. The final schedule for this simplified example would be:

Time	Task	Qty	Time spent	Time available
7 PM	Rebandage fox leg wound	-	20	40
	Feed kits	-	30	10
	Feed porcupines	2	10	0
8 PM	Feed coyotes	8	50	10
	Feed porcupines	2	10	0
9 PM	Feed kits	-	30	30
	Feed porcupine	1	5	25

You are being asked to design an application which will calculate a schedule for an entire day. EWR currently uses a database which is updated throughout the day by the staff vet to keep track of required medical tasks. Each animal will have at least 1 medical task associated with it. Your application should read from the same database to identify the tasks. EWR expects that Sara will run the program each evening and, if necessary, contact the backup volunteer.

Feeding information is not included in the database. It is based on set times according to whether the animal is nocturnal, diurnal, or crepuscular. The amount of time required for feeding depends on the type of animal (e.g., coyotes always take 5 minutes per animal, with 10 minutes of preparation time for any feeding activity). For orphaned animals, feeding is considered a medical task rather than a standard task. Cage cleaning is another activity not included in the database. It can be done at any time of day, and takes a consistent amount of time for each type of animal.

The program should create a file containing the schedule. If Sara needs to call the backup volunteer, the program should inform her and it should not exit until she has confirmed that she has called the volunteer for the necessary time(s). If it is not possible to create a schedule, the program should report the problem so that Sara can contact the staff vet to request a change of requirements. As the staff vet may not be on site, Sara needs to be able to modify the start hour of one or more of the treatments based on the advice she receives. The new schedule should be created after the database is updated.

Design Specifications

You can assume that:

- The only types of animals cared for are those specified in this list. The only values in the associated database field will be in lower-case and in singular form: 'coyote', 'fox', 'porcupine', 'raccoon' or 'beaver'.
- Each cage needs to be cleaned once per day.
- Orphaned animals of the same litter share a cage and are listed as a single animal in the database.
- Nocturnal animals are fed in a 3-hour window starting at midnight (0). That is, feeding is scheduled for 12 AM, 1 AM, or 2 AM.
- Diurnal animals are fed in a 3-hour window starting at 8 AM (8).
- Crepuscular animals are fed in a 3-hour window starting at 7 PM (19).
- Coyotes are crepuscular. Each takes 5 minutes to feed and the feeding preparation time is 10 minutes. It takes 5 minutes to clean each cage.
- Porcupines are crepuscular. Each takes 5 minutes to feed with no preparation time. It takes 10 minutes to clean each cage.
- Foxes are nocturnal. Each takes 5 minutes to feed and the feeding preparation time is 5 minutes. It takes 5 minutes to clean each cage.
- Raccoons are nocturnal. Each takes 5 minutes to feed with no preparation time. It takes 5 minutes to clean each cage.
- Beavers are diurnal. Each takes 5 minutes to feed with no preparation time. It takes 5 minutes to clean each cage.

EWR has provided a sample database showing medical treatments for the animals included in Sara's example. Note that her example is only for part of the day, while the medical treatments in the file cover the full day. The provided file is `treatments.sql`

The full list of tasks shown in the database is:

- Feed Annie, Oliver and Mowgli at 0 AM or 1 AM, at 2 AM or 3 AM, at 4 AM or 5 AM, at 6 AM or 7 AM, at 8 AM or 9 AM, at 10 AM or 11 AM, at 12 PM or 1 PM, at 2 PM or 3 PM, at 4 PM or 5 PM, at 6 PM or 7 PM, at 8 PM or 9 PM, and at 10 PM or 11 PM. Each feeding takes 30 minutes.
- Give Loner eyedrops at 10 PM. It will take 25 minutes.
- Inspect Biter's broken leg. It will take 5 minutes and should be done at 1 PM or 2 PM. Give Biter eyedrops, which will take 25 minutes and should be done at 1 PM.
- Bitter gets a vitamin injection at 1 PM, 2 PM, 3 PM, 4 PM or 5 PM. It takes 5 minutes.
- Pencil gets a vitamin injection at 1 PM, 2 PM, 3 PM, 4 PM or 5 PM. It takes 5 minutes.
- Eraser has a neck wound which is flushed at 1 PM. It takes 25 minutes.
- Slinky has a leg wound rebandaged at 7 PM. It takes 20 minutes.
- Spike has a neck wound which is flushed at 6 AM. It takes 25 minutes. Spike gets antibiotics at 6 AM and 6 PM. It takes 5 minutes each time.
- Javelin needs eyedrops at 10 PM. This will take 25 minutes.
- Gatekeeper gets a vitamin injection at 11 PM, 12 AM, 1 AM, 2 AM, or 3 AM. It takes 5 minutes.
- Sunshine gets mange treatment at 11 PM, 12 AM, 1 AM, or 2 AM. It takes 15 minutes.
- Shadow gets burn ointment at 11 PM, 12 AM, or 1 AM. It takes 10 minutes.
- Boots gets a fluid injection at 10 PM. It takes 10 minutes.
- Spin receives mange treatment at 1 PM, 2 PM, 3 PM or 4 PM. It takes 15 minutes.
- Spot receives mange treatment at 1 PM, 2 PM, 3 PM or 4 PM. It takes 15 minutes.

Additionally, the following tasks can be inferred:

- Feed Loner, Biter, Bitter, Pencil, Eraser, Boots, Spin and Spot once a day. It needs to be done at 7 PM, 8 PM, or 9 PM and takes 5 minutes per animal, with 10 minutes of preparation time per hour. Their cages need to be cleaned daily; it takes 5 minutes per animal.
- Feed Slinky once a day, at midnight, 1 AM, or 2 AM. It takes 5 minutes to feed Slinky, and 5 minutes of preparation time.
- There are two fox cages to clean, one for Slinky and one for Annie, Oliver and Mowgli. It takes 5 minutes to clean each cage.
- Feed Spike, Javelin, Gatekeeper, Sunshine and Shadow between 7 PM, 8 PM and 9 PM. It takes 5 minutes to feed each animal. Their cages need to be cleaned once per day, and it takes 10 minutes to clean each cage.

You can assume that we will use different data in each database table when your program is tested.

The schedule that you write must include the following information:

- The hour that the task should be performed
- The tasks to be performed in the hour
- If the backup volunteer is required

You may choose your own formatting, wording, etc. A schedule showing Sara's example is provided in `schedule.txt`

The program should also display all scheduling information and require a confirmation from the user for each instance that the backup volunteer needs to be contacted.

Technical Specifications

- All classes should be public, with access to functionality provided through public methods.
- Each method should have discrete functionality, be free of side effects, and designed to facilitate unit testing, even if it is a private method. Some guidelines:
    - Methods exceeding 24 lines in length should be examined carefully to ensure that they adhere to all the rules.
    - Ensure that the name of the method clearly indicates all expected behavior (to prevent side effects) - if the name is too long, the method is probably not discrete.
    - Consider the different branches that varied input could take through the method. Branches grow exponentially: a single if/else statement has 2 possible branches, but a method with 4 if/else statements has 16 (2*2*2*2). Imagine writing a code for each branch, and if the number is overwhelming, think of how you could break it into smaller pieces: 4 methods, each with one if/else statement, requires only 8 (2+2+2+2) tests to evaluate every branch, compared to the 16 required for one method with 4 if/else statements.
- You may assume that the database data is correct and has already been vetted for invalid inputs, but it cannot be assumed that you will always be able to successfully connect to the database or write to a file, and the code must respond accordingly. However, tests do not need to be written to cover these scenarios.

- Your code should not rely on any packages outside of the `java.*` namespace, with the exception of `junit` and `hamcrest.core`. You cannot expect TAs to install any new software to run your code.
- The program should not exit without generating a schedule and getting confirmation for all backup volunteers (i.e., all errors should be caught before they are thrown by the JVM).
    - Users should be provided with meaningful error messages appropriate to the audience (end user, rather than programmer). Error messages must explain what was wrong and how it can be corrected (e.g., "It was impossible to complete the schedule due to too many events scheduled at 11 AM. Please shift some of the following activities: rebandage head wound (Spike), administer antibiotics (Spike, Shadow)."
    - Users should be given the opportunity to correct their mistakes, and opportunities for mistakes should be limited. Some techniques for reducing user error are:
        - Providing instructions prior to data entry
        - Normalization of input
        - Limiting choices to only valid options

Deliverable Descriptions

Early UML Draft

Create a legible UML diagram that accurately represents the object-oriented design of your application. You do not need to include your main or GUI interaction components. You may also assume that any database connection/importing will be added later. You will receive feedback on your OOP design - your classes should work together through relationships rather than being a linear sequence of functionality.

When submitting this deliverable, you must include a submission comment with the URL to your main repository.

After receiving feedback, you may modify your UML diagram as necessary before resubmitting as part of the final package.

Test Design

Create a JUnit 4 test suite that can be used to assess the success of your code. You should consider as many boundary cases as possible. You will receive feedback on the thoroughness of your tests and your consideration boundaries, invalid data, exceptions, etc. Your tests will not be run against your code at this point, as your code will still be in development. Instead, it will be compared against your UML diagram and the specifications. If your UML diagram has changed since the first milestone, you can include the updated diagram. Be sure to document your tests well.

After receiving feedback, you may modify your tests as necessary before resubmitting as part of the final package.

Final Package

Your final package will consist of several components to present your solution.

All of the deliverables should be inside a single .zip folder labelled with your group number. Only one team member needs to upload the solution to D2L. You must include all of your team members' names on each deliverable (i.e. in the code documentation, unit tests, etc.).

Upon submission, you should also invite `abarcomb-work` (ENSF 380 and ENSF 409) and `mostafama` (ENSF 380 only) to contribute to your repository. We will not modify your repository but may use the access to review team contributions.

UML Diagram (15%)

You must submit a UML diagram that accurately represents your final code design. The diagram must be legible and follow proper formatting conventions. You do need to represent exception handling in your diagram.

Code implementation (40%)

All of your solution's .java files should be included. You should use the edu.ucalgary.oop package. Each public class must be placed in its own file.

Your code should be developed using GitHub. Each team member should fork the repository and use pull requests, as this will confirm code contributions in case of team disputes. Suggested workflow:

- Have one person create the project on GitHub. Make sure that the repository is private, and access is granted to team members. People who are not part of the project team or the teaching team should not be able to view the code.
- Each person, including the one who created the project, should fork the repository and work on their fork, sending pull requests to the original project. While we will not enforce this model of working, do note that it is in your interests to follow it, as it ensures that (1) you have a copy of the code for future reference or inclusion in portfolios (in the event of the project owner removing you or deleting the project), and (2) there is a record of your actual contributions in case there is a subsequent dispute about the contributions made by each group member.
- If you pair program, take turns making commits so that there is a record of your contributions.
- Your final code will be submitted via D2L dropbox.
- In the case of any disputes about contributions, we will review your GitHub repository and log. It is to your advantage to ensure that the history accurately reflects your contributions.

Documentation

All of your code should be fully documented and commented. Formatting and naming conventions should follow the standards used in the course. Your main should include instructions for running and using your code. You may also choose to include a README text file instead. It is expected that your documentation is sufficient for both a user (README or main) as well as for a programmer who wishes to extend the program.

Unit Tests

You must include unit tests that demonstrate how your solution meets the design specifications. Consider possible boundary cases and exception handling. Your code will be run against your own tests, though values may be changed to ensure that solutions are not hard-coded to specific scenarios.

Video demonstration

Use Zoom or another tool of your choosing to record a short demonstration of your application (5 minutes or less!). All team members should be part of the demonstration. Marks may also be deducted for videos of excessive length. Videos should have clear audio and cameras should be on or you should display a professional-looking photo. This is your opportunity to demonstrate how your application works and why it fulfills the requirements.

Grading Rubric

Each deliverable will be graded using a rubric. You must satisfy all of the criteria in a particular category to earn that category's minimum grade. If you meet the majority, but not all of the criteria for the next category, an intermediate grade may be given (e.g., all requirements for 80% are met, but 2 requirements for 90% are missing - the grade is 85%). Grades will be multiples of 5 (e.g., 80%, 85%, 90%). The grading criteria can be found in the accompanying rubric handout.

Team Assessment

Following the project submission, you will have the opportunity to reflect on the contributions of each team member. You will allocate a percentage of work to each member. For example, in a team of four, if all members contributed equally, you would allocate 25% to each person. If one student took on more responsibilities compared to the others, the allocation might be adjusted to 40%, 20%, 20%, 20%. These allocations may be used to adjust your individual grade on the project in case of uneven contributions. Any discrepancies between allocations will be flagged for the TAs and instructors to investigate further. If you do not submit an evaluation, it will be assumed that you believe the work was evenly distributed.

If each team member contributed equally to the project, no change will be applied the your grade. If the contributions were weighted unequally, your project grade will be individually weighted according to the assessment average.

Actual/Expected * Team Eval Average (maximum weighting of 1.25 and you cannot earn more than 100%)

An even distribution has a weighting of 1.

For example:

Team A of four students received 75/100. Average peer assessment of contributions was 25%, 25%, 25%, 25%. All team members receive 75/100 * 1 = 75%.

Team B of four students received 75/100. Average peer assessment of contributions was 50%, 10%, 20%, 20%.

- 50% is an increase of 25% in the contribution weighting so Member 1 receives 75/100 * (1 + 0.25) = 94%
- 10% is a decrease of 15% in the contribution weighting, so Member 2 receives 75/100 * (1 – 0.15) = 64%
- 20% is a decrease of 5% in the contribution weighting, so Member 3 receives 75/100 * 0.95 = 71%
- 20% is a decrease of 5% in the contribution weighting, so Member 4 receives 75/100 * 0.95 = 71%

Team C of three students received 75/100. Average peer assessment of contributions was 33%, 33%, 33%. All team members receive 75/100 * 1 = 75%.

Team D of three students received 75/100. Average peer assessment of contributions was 90%, 10%, 0%.

- 90% is an increase of 57% in the contribution weighting, but the max is 1.25, so Member 1 receives 75/100 * 1.25 = 94%
- 10% is a decrease of 23% in the contribution weighting, so Member 2 receives 75/100 * 0.77 = 58%
- Students who do not contribute any work will receive zero, so Member 3 receives 75/100 * 0 = 0%

Note: any large standard deviations among peer assessments will be investigated further (looking at your git commits). The teaching team will then make an assessment based on their estimation of the amount of work each member.