Project 5 - Pokédex

CSC-121, Fall 2025

[A Letter from Professor Oak](#)
[Overview](#)
[Getting Started](#)
[Asset Data](#)
- [Pokemon Data CSV](#)
- [Images](#)

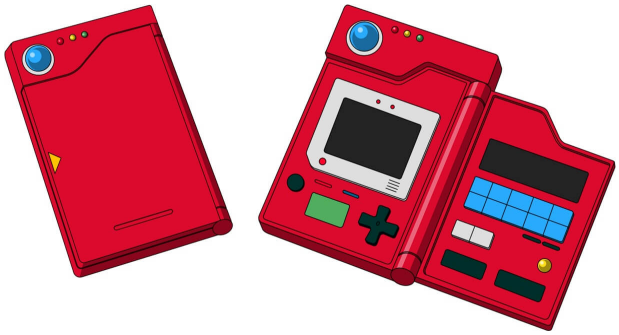
[Student Requirements and Tasks](#)
[Exporting an Executable JAR File](#)
[Notes on genAI usage:](#)
[Explain your work](#)
- [Manual](#)
- [Video](#)

[Grading - Deliverables and Evaluation Criteria](#)
- [Checkpoint (10%)](#)
- [Video Explanation (20%)](#)
- [Code submission (60%)](#)
- [Additional Documents (10%)](#)
- [Late Policy](#)

A Letter from Professor Oak

"Hello there! Welcome to the world of POKÉMON! My name is Professor Oak! People call me *The Pokémon Prof!* This world is inhabited by creatures called **Pokémon**! For some people, Pokémon are pets. Others use them for fights. Myself… I study Pokémon as a profession."

"Oh, right! I have a request of you. On the desk here is my invention: The PokéDex! It automatically records data on Pokémon you've seen or caught! It's a hi-tech encyclopedia!"



"…Or at least, *it could be*, but I need the help of a talented software engineer to finish programming it. To make a complete guide on all the Pokémon in the world… That was my dream! But I'm too old! I can't do it! So, ***I want you to fulfill my dream for me!*** Get moving!"

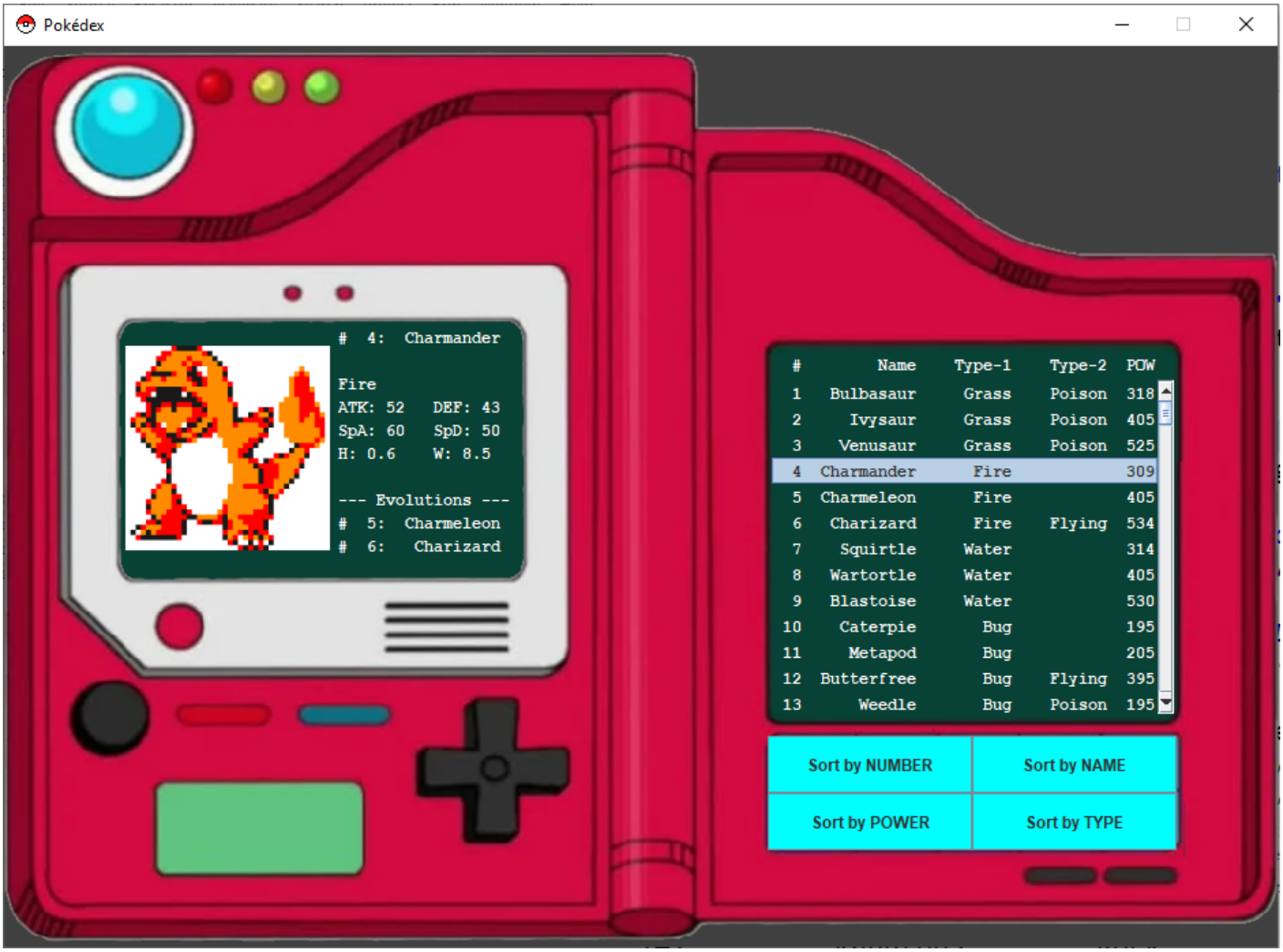
"Your very own Pokémon legend is about to unfold! A world of dreams and adventures with Pokémon awaits! Let's go!"

- *Professor Oak*

Overview

Pokémon are the cute and powerful creatures in the Pokémon games and anime series. Trainers catch wild Pokémon and train them to do various things. Some Pokémon are intelligent, some can be ridden to cross terrain, some are fierce fighters, and many can grow and evolve into new, stronger forms.

The Pokédex is an electronic personal assistant and encyclopedia that delivers helpful information to Pokémon trainers about the behaviors, abilities, and evolutions of Pokémon. In this final project, you will be finishing the development of Professor Oak's Pokédex.



Above: The Pokédex, in Java.

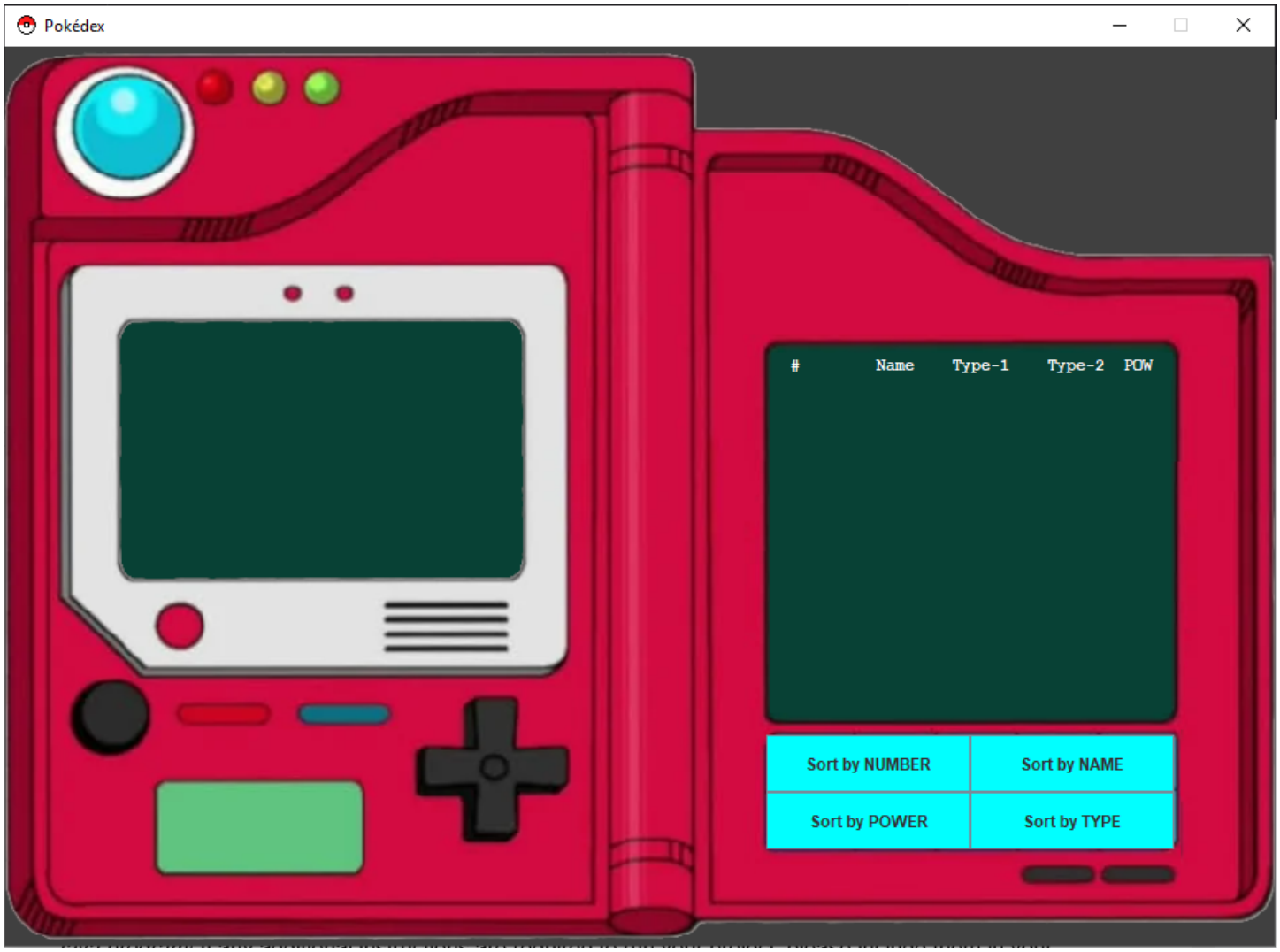
Our Pokédex is a standalone Java application that uses Java's built-in Swing and AWT libraries to make a GUI (Graphical User Interface). The right-side of the Pokédex application shows a scrollable list of all known Pokémon. The list can be sorted in several different ways by pressing the buttons below the screen. When an entry in the list is clicked, information about that Pokémon and an image of them is displayed on the left-hand side of the application. You can see the Pokemon's name and ID number, its elemental type, power levels, and the new forms the Pokémon may evolve into as it grows.

The majority of the GUI features have been implemented for you. Your job is to load the information on all of the Pokémon into the Pokédex, format the information to display well on the device, implement the sorting algorithms, and write the evolutionary growth tracking.

Getting Started

Download the starter project and import it into Eclipse. We *strongly recommend* using the starter project rather than trying to build the entire project from scratch on your own. There are walk-through guides on importing projects into Eclipse and Visual Studio Code on Canvas that you can use. You can also ask a TA to help if you have any trouble.

The starter project has no errors, so you should be able to run the project as soon as you import it. Below is what you should see if everything is working correctly with the starter code.



Above: The starter project.

Asset Data

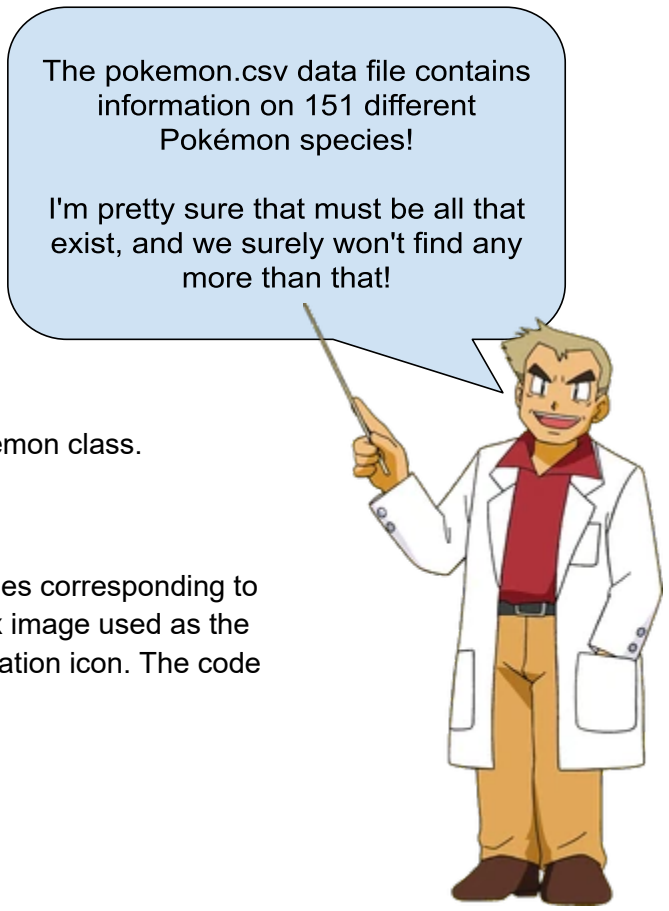
Data files for the project are located in the "data" directory in the project root. Let's take a look at the contents of this folder.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	#	Name	Type 1	Type 2	Evolution	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Power	Height	Weight	Catch Rate
2	1	Bulbasaur	Grass	Poison	2	45	49	49	65	65	45	318	0.7	6.9	45
3	2	Ivysaur	Grass	Poison	3	60	62	63	80	80	60	405	1	13	45
4	3	Venusaur	Grass	Poison	0	80	82	83	100	100	80	525	2	100	45
5	4	Charmander	Fire		5	39	52	43	60	50	65	309	0.6	8.5	45
6	5	Charmeleon	Fire		6	58	64	58	80	65	80	405	1.1	19	45
7	6	Charizard	Fire	Flying	0	78	84	78	109	85	100	534	1.7	90.5	45
8	7	Squirtle	Water		8	44	48	65	50	64	43	314	0.5	9	45
9	8	Wartortle	Water		9	59	63	80	65	80	58	405	1	22.5	45
10	9	Blastoise	Water		0	79	83	100	85	105	78	530	1.6	85.5	45
11	10	Caterpie	Bug		11	45	30	35	20	20	45	195	0.3	2.9	255
12	11	Metapod	Bug		12	50	20	55	25	25	30	205	0.7	9.9	120
13	12	Butterfree	Bug	Flying	0	60	45	50	90	80	70	395	1.1	32	45
14	13	Weedle	Bug	Poison	14	40	35	30	20	20	50	195	0.3	3.2	255
15	14	Kakuna	Bug	Poison	15	45	25	50	25	25	35	205	0.6	10	120
16	15	Beedrill	Bug	Poison	0	65	90	40	45	80	75	395	1	29.5	45

Above: A snapshot of the pokemon.csv data file.

Pokemon Data CSV

The data for all of the Pokémon are stored in a CSV file named "*pokemon.csv*". Each row of the CSV represents the information of a single Pokémon. The columns represent individual properties of each Pokémon, including their ID number, name, elemental types (All Pokémon have at least one type, and some have two types), the ID number of another Pokémon that this Pokémon can evolve into (Or 0 if this Pokémon does not evolve), fighting statistics (HP/Health Points, Attack, Defense, Special Attack, Special Defense, and Speed), a power rating (The value of all the fighting statistics added together, higher number means stronger overall stats), height in meters, weight in pounds, and the catch rate (How difficult it is for a trainer to catch this Pokémon in the wild, with higher numbers being easier to catch and lower numbers being harder to catch). This CSV should be read when the program starts and stored in instances of the Pokemon class.



Images

Inside the data/images directory are 151 numbered image files corresponding to the ID numbers of all the Pokémon. There is also a Pokédex image used as the GUI background, and a Pokéball used for the window application icon. The code to load these assets is already provided for you.

Student Requirements and Tasks

Students must download the project framework and import it into Eclipse as a starting point. All of the student work will be within the Pokemon.java and Pokedex.java files. The BackgroundPanel.java file is included for some additional UI functionality and does not require any student work. Also, the top-half of Pokedex.java contains Java UI code which is already completed. Sections requiring student implementation are in the bottom-half of the file. Below is a rough outline of the required tasks. Use Ctrl+F to search through the Pokemon.java and Pokedex.java files for `TODO` notes which also hint at where student work is needed.

☐ A. Complete the Pokemon Class

The Pokemon class should hold all of the data on an individual Pokémon species. Read in each row of the pokemon.csv and create an instance of the Pokemon class for each row. The Pokemon class should hold a private instance field for each column of information. Write a constructor that takes in the row data and initializes the object's fields. Look through the class for other `TODO` notes that mark tasks that should be completed. Most of them involve print formatting. Also write any other methods and getters/setters that you feel like you need. You do not need to write getters and setters for every field if you do not think that you need them - just write the methods you deem *useful*.

NOTE: We have been using the accented é a lot in this document, but you may notice we do not use it when referring to the words Pokemon or Pokedex in code. Be very, very careful about writing accented character symbols as code syntax. It may cause odd, difficult to find bugs.

☐ B. Load pokemon.csv Data File

Load the CSV file in the Pokedex class' loadData() method. You should do the following:

a. Create an instance of Pokemon for each row in the CSV.
b. Add the instance to both the *Pokedex.pokemon* and *Pokedex.orderedPokemon* ArrayLists.
c. Print out each of your Pokemon objects using the Pokemon.toString() method.

When you complete tasks A and B, the list on the right side of the Pokédex should populate.

☐ C. Implement the Four Sorting Methods

The Pokédex can sort the list of pokemon on the right by ID number, name, type, or power rating. Each sorting mode has a corresponding method: Pokedex.sortByNumber(), Pokedex.sortByName(), Pokedex.sortByType(), Pokedex.sortByPower(). You may implement the four methods using any sorting algorithms you wish, though you must follow these rules:

a. You may not use any built-in sorting methods like the Java Collections.sort methods. You must write the sorting algorithms yourself.
b. You must implement at least one of the methods with a Bubble Sort.
c. You must implement at least one of the methods with a Selection Sort.
d. You *may* choose to use the same sorting algorithm for more than one method if you like. So, you *could* write the four sorts using two selection sorts and two bubble sorts and call it a day.

☐ D. Implement a Binary Search for Pokedex.findPokemonByNumber()

Several parts of the program need to find a Pokemon object by its ID number. The method Pokedex.findPokemonByNumber() is meant to do this for us. To make it fast, implement the method using a binary search to find the target Pokemon in the Pokedex.orderedPokemon ArrayList.

NOTE 1: Implementing a sequential search instead of a binary search will result in point penalties.

NOTE 2: Yes, we *could* just get the Pokemon instance directly from the sorted ArrayList with its ID number as an index, ***but where's the fun in that?***

☐ E. Recursively find Pokemon Evolutions

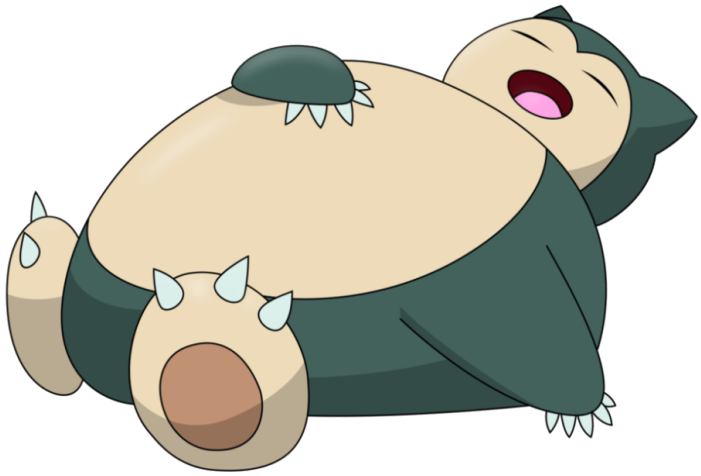
We want to display all of the evolutionary forms a Pokémon can evolve into over time in the detailed view on the right, but each Pokemon instance has at most one evolution stat. This stat lists the ID number of the next Pokémon form in line. To find the next form after that, you must use the evolution stat of the new form, and so on. Implement the recursive logic in the method Pokedex.findEvolutionsRecursively(). Add all of the forms to the "evolutions" ArrayList.

NOTE 1: Using iterative logic for this step instead of recursive logic will result in point penalties.

NOTE 2: Sharp-eyed Pokéfans may spot that the Eevee evolution information is not correct in the CSV data. Our current system does not support Eevee's unique evolution branching behavior, so we've deliberately listed Eevee's evolution information incorrectly to make implementation easier.

☐ F. Create an Executable JAR File

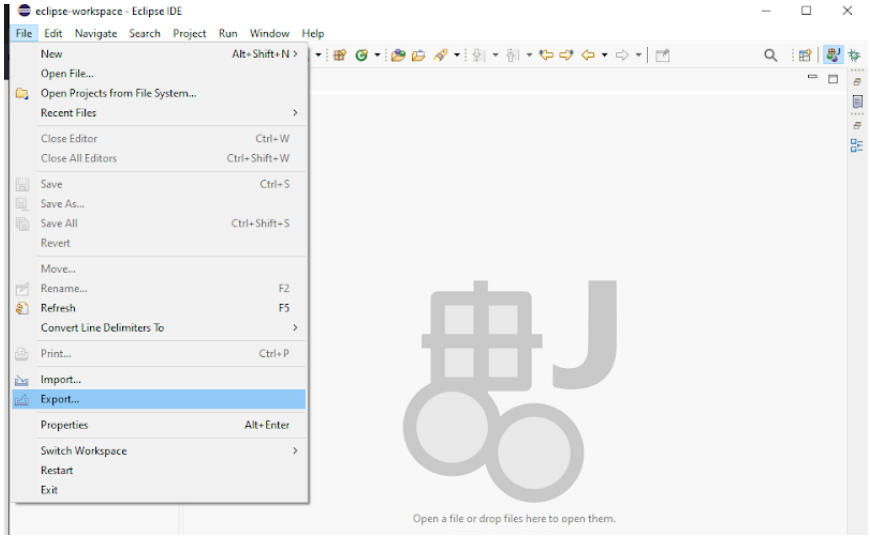
In addition to your code, manual, and video, export your project as an executable JAR file from Eclipse. Ensure that you can run the JAR on your own local machine after you create it. Submit your JAR file along with your other project files.



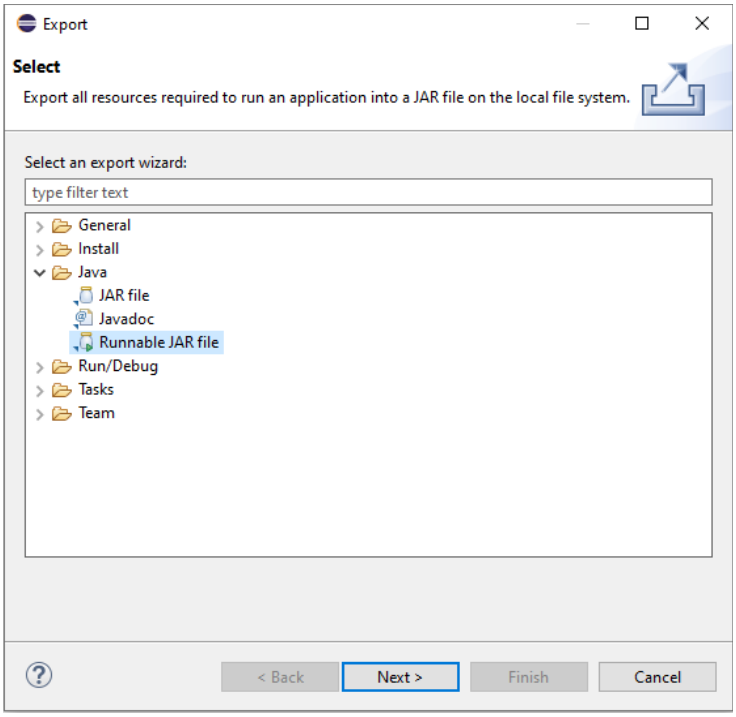
Exporting an Executable JAR File

It's nice that we can run our code in Eclipse, but it would be nice if we could package up our program and give it to other people. Java lets us do that by bundling our bytecode up into a JAR file. Anyone with a Java runtime environment can run our program from the JAR.

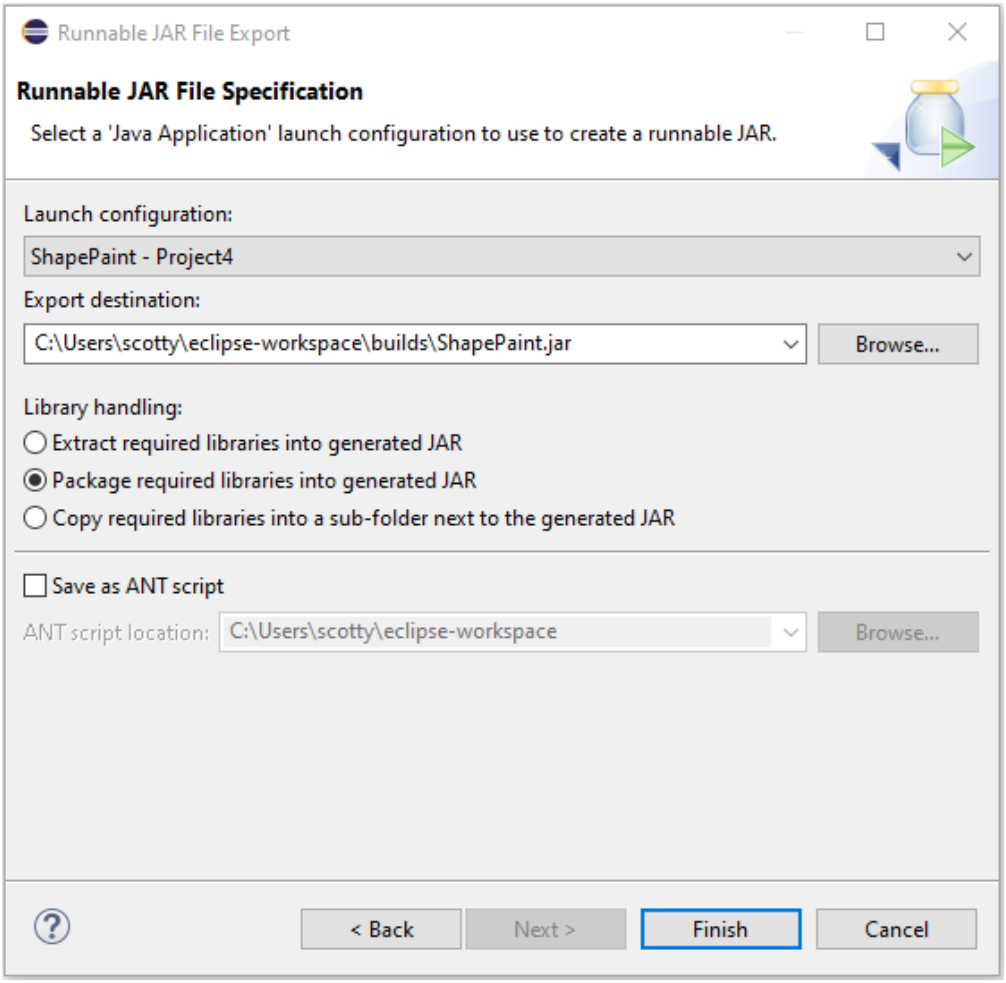
1. In Eclipse, select "File -> Export".



2. For the export type, select "Java -> Runnable JAR file".



3. For the "Launch configuration", select your project and the class that contains your "public static void main" method (For this project, it should be something like "ShapePaint - Project4"). Choose a location to export the JAR file to on the "Export destination" box. Under "Library handling", click "Package required libraries into generated JAR". Then click Finish.



4. Test your JAR file by double-clicking it and see if your program launches. Depending on your operating system, you may need to give the JAR file executable privileges before it will run. Then you are ready to share your program with the world!

~~ IMPORTANT!! ~~

5. To run your JAR, you will need to have a copy of the data directory in the same directory as your JAR file. If it is missing, you will notice missing images and no CSV data. You do NOT need to submit the data folder with your project submission, though.

Notes on genAI usage:

You are encouraged to use an AI tool while writing your code. For example, you may want to ask it to explain a concept related to your program, such as what operator to use to check if two values are equal. Or, if you find you have a syntax error, you could leverage genAI as part of your debugging process. However, you should not trust AI tools blindly, or copy and paste AI-generated code without understanding how it works. ***You should probably avoid using concepts that we have not yet covered in class, even if AI suggests them***. Keep in mind that you will need to explain your code on video, and must understand your code to do so.

Explain your work

Manual

In addition to your project code .java files and video, you must write a short manual to accompany your project. Name your plain text manual "README.txt". Include any information you deem necessary to running and using the program. You should also mention the names of your favorite Pokémon. (Personally, I'm quite fond of Cubone, Porygon, Snorlax, and Jigglypuff.)

Video

A final part of the project is to explain your work and demonstrate your understanding of the program you created.

You will share your explanation in a 5-minute video after creating a detailed outline of the points that you want to make. Specifically, you must explain the following:

- The intended functionality of this program.
- Your process and the steps you took to design and develop this program.
- How the code executes in one of your most complex sections.
- The lessons you learned through this project

Each part should be at least 1-minute long. You can use a standard cellphone or webcam to record your video. If you use a screen recording program, we still need to be able to hear your voice explaining the code and project.

Please submit your video using a standard video format, like .mp4 format. If you are unable to submit the video file along with your submission on canvas for any reason, you may instead upload the video to a free service such as YouTube or Vimeo and include a URL link to the video with your project submission in a plain text file named "video.txt". Your video must remain accessible to grading staff until grading is complete.

Grading - Deliverables and Evaluation Criteria

You will be responsible for the following deliverables. The weighting of each component is indicated below. It's important that you refer to the following Project Rubric for detailed information about how each deliverable will be evaluated.

Checkpoint (10%)

Each student must meet with the professor or a TF in an in-person meeting before the project deadline to discuss their project, and any problems they may be encountering. Feedback will be offered to help students with building their project. Be sure to ask the professor/TF to count your project check-in.

Video Explanation (20%)

You must submit your video (Or a plain text file with a link to your video) containing your explanation along with your detailed outline by the due date.

Code submission (60%)

You must submit your code by the due date. *Please ensure it is possible to successfully compile and run your Java program!* If any additional instructions are required to run your project, please include them in your manual.

Additional Documents (10%)

- Manual
  You must submit a plain text manual file explaining how to use your program. This should cover an overview of what the program is, and any special instructions for running or using the program (Such as keyboard controls) and the names of your favorite Pokémon.

- Executable JAR
  Export your project as an executable JAR file from eclipse. Instructions for creating an executable JAR are included in this document. Ensure that you can run the JAR on your own machine after you create it. Submit your JAR file along with your other project files.(You do NOT need to submit the data folder.)

Late Policy

Projects submitted after the deadline will be subject to a 20% late penalty.

PROTIP! Detective Pikachu Says:

Don't wait until the last minute to start the project!

Start early, and ask for help if you get stuck!