WIX1002 FUNDAMENTALS OF PROGRAMMING

General Assignment Instructions

This document provides all the general guidelines, submission requirements, and reporting standards for your WIX1002 project.

1 Project Submission

- All submissions must be made via the **GitHub Classroom repository** assigned during your class.
- Your repository must contain all your Java source code.
- A complete project report must be included in your `Readme.md` file (see requirements below).

2 Project Report (Readme.md Requirements)

The `Readme.md` file in your group's GitHub repository serves as your **official project report**. It must be professional, clear, and provide a complete overview of your project.

Your `Readme.md` **must** include the following sections:

2.1 Project Title

- The full title of your project (e.g., `WIX1002 FUNDAMENTALS OF PROGRAMMING - Topic 1: Planet Explorer Game`).

2.2 Team Members

- A numbered list of all group members, including their full names and student IDs.
    - `1. Full Name (Student ID)`
    - `2. Full Name (Student ID)`

2.3 Project Overview

- A brief introduction to your project.
- This section should paraphrase the "Overview" and "Problem Statement" from your topic sheet, explaining the purpose and main goal of your application.

2.4 Features Implemented

This is a critical section for grading. You must clearly state which features you completed.

- **Basic Features (8 marks):**
    - List every basic feature from the assignment sheet.
    - Mark each feature as **"Completed"**, **"Partially Completed"**, or **"Not Implemented"**.

    - Provide a brief description of *how* it was implemented (e.g., "Authentication: Completed. User data is saved in a text file.").
- **Extra Features (max 4 marks):**
    - List any extra features you added.
    - Explain what each feature does and why you believe it merits extra marks.

2.5 How to Compile and Run

Provide clear, step-by-step instructions for your lecturer or demonstrator to compile and run your project.

- **Dependencies:** List any special libraries required (e.g., "Requires JavaFX 11").
- **Compilation:** Provide the exact command(s) to compile your code (e.g., `javac *.java`).
- **Execution:** Provide the exact command(s) to run the application (e.g., `java MainMenu` or `java -jar Project.jar`).

2.6 User Guide / How to Play

Explain the controls and rules of your application/game.

- Example:
    - `Use WASD or Arrow Keys to move the spaceship.`
    - `Press 'P' to pause and resume the game.`
    - `Collect resources to score points; avoid asteroids.`

2.7 Screenshots

- Embed screenshots directly into your `Readme.md` to showcase your work.
- You must include:
    - The main menu / login screen.
    - The main gameplay / application screen.
    - The leaderboard or high score screen (if applicable).

2.8 Contribution

Since collaboration is being evaluated, you must detail the contribution of each team member.

- Be specific about which member was responsible for which features or components.
- Example:
    - **Member 1:** Authentication, GUI, and Player Spaceship.
    - **Member 2:** Obstacles, Collision Detection, and Scoring System.

2.9 Challenges Faced (Optional)

- A short paragraph describing any difficult technical problems you solved.
- This section demonstrates your learning process (e.g., "We found it challenging to implement smooth collision detection, but we solved it by...").

3 General Guidelines

(a) **Group Size:** Each project group must consist of **no more than 5 students**.
(b) **GitHub Requirement:** You are **required** to use GitHub for version control and collaboration, as this facilitates effective teamwork.
(c) **Collaboration Workflow:** Teamwork proficiency will be assessed. You **must** adopt a workflow utilizing **Pull Requests (PRs)** for all code contributions. This process, including code review and discussion, may be evaluated.
(d) **Teamwork Issues:** Any issues regarding team collaboration (e.g., non-participating members) must be reported to your tutor, as teamwork will impact the final grade.
(e) **Data & File I/O:** Pay careful attention to data input and file manipulation (e.g., CSV, JSON, images). This is a critical component, and errors in data handling can lead to significant issues.
(f) **File Paths:** You **must use relative file paths**. Absolute file paths (e.g., `C:\Users\user...`) are not permitted and will result in marks being deducted, as they prevent the code from running on other machines.
(g) **Code Quality & Readability:** Your code must be clean, well-commented, and readable. Use clear and conventional naming for all variables, methods, and objects. This is essential for both collaboration and grading.
(h) **Standardized Development Environment (Recommended):** To prevent common "it works on my machine" problems, you are encouraged to use a VS Code Dev Container. This ensures every team member and your tutor has the exact same Java version and libraries. A basic `.devcontainer` configuration may be provided in your GitHub Classroom repository to simplify setup.

4 Application Output

(a) All projects must produce clear, usable output. The implementation may be a Command-Line Interface (CLI) or a Graphical User Interface (GUI).
(b) **For non-GUI (CLI) applications:** Output must be well-formatted, with clear labels and menus to ensure the application is user-friendly.
(c) **For GUI applications:** The interface must be intuitive, responsive, and visually organized.
(d) **Note:** Refer to your specific topic file. Some topics may **require** a GUI, while others may award **extra marks** for one.
(e) Regardless of the approach, screenshots of the final application running are a **mandatory** part of your `Readme.md` report.

WIX1002 FUNDAMENTALS OF PROGRAMMING

Topic 1: Planet Explorer Game

Overview

This project requires students to design and implement an interactive Planet Explorer Game using Java. The application simulates a player controlling a spaceship to explore various planets, collect resources, and evade asteroids and enemy ships. As resources are collected, the player unlocks upgrades for their spaceship and progresses through four difficulty levels.

Students may implement additional features, such as a leaderboard, sound effects, high score display, and secure user authentication, for extra credit.

Problem Statement

Design and develop a Planet Explorer Game in Java that incorporates fundamental gameplay mechanics. In the game, the player controls a spaceship that collects resources scattered across various planets to enhance its capabilities.

While exploring, players must carefully avoid collisions with asteroids and enemy ships, as these can damage the spaceship or end the game. Players earn points for collecting resources, unlock upgrades as they progress, and strive to achieve the highest possible score. The game should feature a graphical interface that allows for smooth and interactive gameplay, and it should include a user authentication system with sign-up, login, and logout functionality to track each player's progress.

Basic Features (8 marks)

Features	Description	Suggestions
Authentication **(1 mark)**	Sign-Up: username, name, password, confirm password; unique usernames. Login: Authenticate returning players and load saved progress. Logout: save progress.	
Graphical User Interface (GUI) **(1 mark)**	Display spaceship, resources, asteroids, and enemy ships. Show score, level, and upgrades. Smooth animations with double buffering.	To create your GUI, consider using libraries like Java Swing, Java AWT, or JavaFX. These libraries offer tools for building windows, buttons, and other interface elements.

		JavaFX is especially useful for building modern and visually appealing GUIs. Refresh Rate & Smooth Animations: For a smooth gaming experience, set a consistent refresh rate. Implement double buffering to reduce flickering and improve performance, ensuring that animations and movements appear fluid.
Obstacles & Enemies **(1 mark)**	Asteroids and enemy ships appear at varying speeds and difficulty per level. Each level introduces stronger or faster obstacles.	Use inheritance to define shared characteristics and behaviours.
Player Spaceship (1 mark)	The player's spaceship can collect resources to grow stronger. Visual upgrades appear as the player levels up.	
Movement **(1 mark)**	Enable the player to control the spaceship using keyboard input.	For example, try using arrow keys, WASD, or a mouse for smooth and responsive movement.
Collision Detection **(1 mark)**	A collision with obstacles can damage or end the game. Collision with resources collects points and upgrades.	
Scoring System (1 mark)	Track points based on resources collected. Display the current score and level on the screen during gameplay for players to track their performance. Display the score and level on the screen after their gameplay.	To save and load player scores, use file I/O methods, such as CSV or text files. For instance, you could store each game session's score and level in a CSV or text file, allowing players to track their progress over time.
Game History (1 mark)	Maintain a record of previous games played, including details such as scores, levels reached, number of gameplays. Allow players to view their game history through the	

	menu, giving them insight into their performance and improvement over time.	

Extra Features (Maximum: 4 marks)

Features	Description	Suggestions
Play/Pause Feature (1 mark)	Allow players to pause and resume the game without losing progress.	
Sound Effects & Background Music (BGM) **(1 mark)**	Add background music and sound effects for eating and collisions.	
Spaceship Customisation / Selection (1 mark)	Provide a menu option to choose from various fish characters with different visual appearances.	
High Score Display (1 mark)	After each game, show the player's highest score on a separate screen or overlay. Allow players to compare their highest score with previous records. Display a congratulatory message when a new high score is achieved. Ensure the high score is also visible during gameplay for ongoing motivation.	If you hit a specific or higher score, you can play with another unique skin of a spaceship.
Leaderboard (1 mark)	Display a leaderboard showing the top scores and ranks of all players.	
Password Hashing (1 mark)	For enhanced security, implement password hashing when storing player passwords in the database. Ensure that during login, hashed passwords are compared to authenticate users securely.	If a user registers, save the password after it has been hashed. When a user wants to log back in, compare the hashed password with the password saved in the local file (Excel, text file, etc.).

WIX1002 FUNDAMENTALS OF PROGRAMMING

Topic 2: Internship Application Tracker

Overview

As Computer Science students approach their final years, securing an internship is a significant challenge. Internships serve as a bridge between academic knowledge and real-world industry experience. However, the application process can be overwhelming, as students often apply to dozens of companies, track multiple statuses (Applied, Interview, Offer, Rejected), and struggle to stay organized.

This project requires students to design and implement an Internship Application Tracker system. This program will allow students to manage all their internship applications in one place, with data stored in a structured JSON file for better organization and scalability.

Problem Statement

Students are tasked to create a program that allows users to organize their internship applications. This system must allow a user to input application information and update that information as it changes.

The user must also be able to view all applications entered into the system and update the status of an application (e.g., set its status to 'Interview'). Finally, the user must be able to delete applications without corrupting the JSON data file.

All internship data must be stored and managed using Java File I/O, where JSON files serve as the main data storage format for reading, writing, and updating application records.

Basic Features (8 marks)

Features	Description	Suggestions
Adding Internship Applications (2 marks)	Users must be able to add applications to their list. All data must be stored in a **JSON file**. The use of text or CSV files is **not allowed**. Users should add: Company Name, Internship Title, Application date, Internship Location (optional), Contact Information (optional), Notes (optional), Internship Status ("Applied" by default). **Conditions:**	Use Object-Oriented Programming (OOP) to encapsulate internship data. Implementing a GUI is highly recommended and will be awarded extra marks. Be cautious with optional details—improperly handled null values can break JSON serialization.

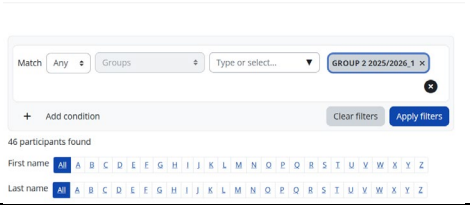
	- Missing Company Name or Internship Title must be rejected. - Application Date must be automatically recorded (system date), not manually entered. - Optional fields left blank should be saved as `null`. - Implement a reasonable word limit (e.g., 50–100 characters) for text fields. - Each record must have a unique ID. - Internship Status should not be entered by the user.	
Displaying Internship Applications (1.5 marks)	Users must be able to view all stored internship applications in a clear and organized format. This view serves as the foundation for update and delete operations.	The display format does not need to be tabular—a neat, list-style view (e.g., card view) is sufficient. Handle `null` values carefully to prevent formatting errors. For GUI users, consider a scrollable list or data table.
Updating / Changing Details (2 marks)	Users must be able to update any details of an internship record (e.g., correct a typo in "Company Name"). **Conditions:** - The same validation rules from "Adding" apply. - Required fields must not be empty. - Word limits must be enforced. All modifications must be written back to the JSON file.	In a GUI, implement an "Edit" button beside each record. In CLI, create a dedicated "Edit" menu. Ensure that updates do not corrupt the JSON format.
Removing Internship Applications (1.5 marks)	Users must be able to delete an internship application. The deletion must be clean, removing the target entry without breaking the JSON file.	Locate the record using its ID, remove it from the applications list, and rewrite the JSON file with the updated list.

		Verify the JSON structure after deletion. In GUI, a "Delete" button is effective. In CLI, use an index or ID prompt.
Marking the Status (1 mark)	Users must be able to change the status of any application. Status changes must be restricted to a predefined set of options: `Applied`, `Interview`, `Awaiting Results`, `Offer Received`, `Rejected`	Use dropdown menus (GUI) or selection menus (CLI) to ensure only valid status values are used. For CLI mode, adding colors for different statuses enhances readability.

Extra Features (Max: 4 marks)

Features	Description	Suggestions
Graphical User Interface (GUI) (2 marks)	Users should be able to interact with the system easily, smoothly, and intuitively. 1 mark for user-friendliness, 1 mark for aesthetic design (beyond default buttons/fonts).	For beginners, utilizing JavaFX or Swing is recommended. For advanced students: React with Java Spring Boot and MongoDB.
Login and Registration (2 marks)	The system can support multiple users with separate accounts. Each user should only see their own internship applications.	Use two JSON files: 1. `users.json` (stores username, password, user_id) 2. `applications.json` (stores applications, each linked to a `user_id`). When a user logs in, the system filters applications by their `user_id`.
Sorting and Filtering (1 mark)	Users should be able to sort or filter applications (e.g., by Date, Alphabetical order). Sort by status (e.g., "Offer Received" at the top). Sort by date applied or company name.	CLI: Add a menu option to choose sorting preference. GUI: Use dropdowns or buttons to sort/filter.

Topic 2: Internship Application Tracker

Search Feature (1 mark)	Users should be able to search applications based on Company Name, Job Title, or Location. 	GUI: Add a search bar at the header for easy access. CLI: Add a dedicated "Search" section in the menu.
Additional Features (1 mark each)	Any other creative features that improve usability or functionality. Examples: Export to CSV, color-code status, reminders for deadlines, dashboard with statistics.	All additional features must work with JSON storage.

WIX1002 FUNDAMENTALS OF PROGRAMMING

Topic 3: FotoEditor

Overview

Digital creativity relies heavily on image editing tools. While professional software exists, it is often inaccessible to students. This project provides an ethical, beginner-friendly alternative for learning image processing and software development fundamentals.

This project requires students to implement FotoEditor, an application that introduces core image manipulation concepts, such as pixel manipulation, color channels, and basic transformations. By building a foundation in these areas, students will gain confidence in developing their own image editing tools.

Problem Statement

Students must design and develop FotoEditor, a Java-based image editing application that allows users to manipulate images at the pixel level.

Each image is to be represented as a 2D array of integers, where each integer encodes the RGBA channels of a pixel in 32-bit format (8 bits per channel). Students must learn to use BufferedImage's getRGB and setRGB methods for pixel-level editing.

The program must allow users to:

(a) Load images.

(b) Apply core image effects (brightness adjustment, blur, monochrome, cropping, resizing).

(c) Optionally, use selective bounding for applying effects to specific regions.

(d) Undo and redo actions.

(e) Save edited images.

All image loading and saving operations must be implemented using Java File I/O methods to handle common formats such as PNG and JPG.

Basic Features (8 marks)

Features	Description	Suggestions
Brightness Adjustment (2 marks)	Develop a feature enabling users to adjust the brightness of an image globally.	Iterate through all pixels and add/subtract intensity values; clamp values between 0–255.
Blur (1 mark)	Implement a blur filter that allows users to apply varying levels of blurriness to an image.	A box blur can be used, where the neighboring pixels' intensity values are averaged.

Monochrome / Grayscale (1 mark)	Convert the image to black and white.	Average the RGB channels or use a weighted luminosity formula: `gray = 0.21*R + 0.72*G + 0.07*B`.
Cropping (1 mark)	Crop the image to a specified rectangular region.	Copy the selected pixel ranges to a new `BufferedImage`.
Resize (1 mark)	Scale the image while maintaining aspect ratio.	Use the nearest-neighbor algorithm for upscaling; average pixels for downscaling.
Undo / Redo (1 mark)	Allow users to undo and redo changes applied to the image.	Maintain stacks of `BufferedImage` snapshots to manage multi-level undo/redo.
Selective Bounding (1 mark)	Apply effects (e.g., Blur/Brightness) to a specific, user-defined area of the image.	Use a rectangle selection tool or a mouse brush for interactive selection in a GUI.

Extra Features (Max: 4 marks)

Features	Description	Suggestions
Color Inversion (1 mark)	Invert the pixel colors to create a "negative" effect.	`newPixel = (alpha << 24) | (~rgb & 0xFFFFFF)`
Sepia Tone (1 mark)	Apply a warm, vintage-style filter to the image.	Use a standard sepia matrix formula for each channel (e.g., `newR = 0.393*R + 0.769*G + 0.189*B`), clamping values between 0–255.
Rotate & Flip (1 mark)	Implement rotation (90°, 180°, 270°) and flipping (horizontal/vertical).	Map original pixel coordinates (x,y) to their new (newX, newY) positions in a new `BufferedImage`.
Draw Shapes / Text (1 mark)	Allow users to add text or basic shapes (rectangles, ovals) onto the image.	Utilize `Graphics2D` methods: `drawRect`, `drawOval`, `drawString`. Stickers could be provided.
Filter Presets (1 mark)	Allow users to save a sequence of applied effects as a reusable preset.	Store the sequence of operations in a list; provide a mechanism to re-apply them to other images.
Additional Features (1 mark each)	Any other creative features that improve usability or functionality.	Examples: Support for multiple image formats, zoom & pan, advanced filters (e.g., seam carving, edge detection).

WIX1002 FUNDAMENTALS OF PROGRAMMING

Topic 4: Travel Planner App

Overview

This project requires students to create an interactive Travel Planner App using Java. The application will allow a user to manage trips and travel activities, including adding tasks, setting priorities, and tracking progress. The system should help users organize trips efficiently while avoiding scheduling conflicts.

Students may implement advanced features such as recurring tasks, analytics dashboards, activity reminders, and secure user authentication for extra credit.

Problem Statement

Students must design and develop a Travel Planner App in Java that incorporates fundamental features of trip and activity management. Users must be able to create trips, add activities with descriptions, dates, priorities, and categories, and manage them effectively.

The system must prevent scheduling conflicts (e.g., overlapping activities) and allow users to mark activities as completed. Users can earn points for completing activities and track their progress. The app should feature a graphical interface for smooth interaction and include a user authentication system with sign-up, login, and logout functionality.

The main data storage for this project will be Excel/CSV files. Database integration is an optional extra feature.

Basic Features (8 marks)

Features	Description	Suggestions
Authentication (1 mark)	Sign-Up: username, name, password, confirm password; unique usernames. Login: authenticate returning players, load saved progress. Logout: save progress.	Save user data in a CSV file. Implement password hashing (e.g., salted SHA-256) for extra security.
Graphical User Interface (GUI) (1 mark)	Provide an interactive dashboard for managing trips and activities. Include input forms, buttons, tables, and progress indicators. Ensure smooth interaction and user-friendly navigation.	Use JavaFX or Swing for building a functional GUI with buttons, forms, and activity lists.
Trip & Activity Creation (1 mark)	Users can create trips and add activities with:	Ensure proper input validation.

	Name, Description, Date, Time, Location, Category, Priority, Completion Status (planned/completed).	Check for overlapping times when creating activities. Allow category selection (e.g., Work, Leisure, Personal).
Activity Management (1 mark)	Users can mark activities as completed or pending. Users can update details (date, description, location). Users can delete activities from trips.	Include clear confirmation messages for updates or deletions.
Activity Sorting (0.5 mark)	Sort activities by date, priority, or location.	Implement a sorting algorithm (e.g., Bubble Sort, Selection Sort) in Java, independent of Excel queries.
Activity Searching (0.5 mark)	Search activities by name, description, location, or time.	Implement linear or binary search in Java. For name-based search, consider ascending/descending options.
Recurring Activities (1 mark)	Allow users to create recurring activities (daily, weekly, monthly). The system should auto-generate the next occurrence after completion.	Provide options to set recurrence when creating an activity.
Activity Dependencies (1 mark)	Allow activities to depend on the completion of others. Warn users if a dependent activity is marked complete prematurely. Prevent cycles in dependencies.	Use algorithms like slow/fast pointers to detect dependency loops.
Storage System (1 mark)	Store all trips and activities in **CSV files**.	Database integration is an extra feature to replace CSV storage. Use relative file paths.

Extra Features (Max: 4 marks)

Features	Description	Suggestions
Play/Pause Scheduler (1 mark)	Pause or resume notifications and reminders for activities.	Useful if the user is away or wants to focus.
Email Notification System (1 mark)	Notify users of upcoming activities (e.g., 24 hours before).	Use JavaMail or another email API. Allow users to configure their email address and notification preferences.

Activity Customization / Tags (1 mark)	Users can categorize activities or tag trips for better organization.	Include filters or color-coding for tags in the GUI.
High Score / Progress Display (1 mark)	Track completion rates and show statistics per trip. Display congratulatory messages for completed trips.	Visualize progress using progress bars or charts.
Leaderboard (1 mark)	Compare completion rates among multiple users.	This is optional for group projects; scores can be stored in a central CSV file or database.
Database Integration (1 mark)	Store all data (trips, activities, users) in a relational database instead of CSV files.	Use JDBC to connect to MySQL, SQLite, or another relational DB. This enables a multi-user storage system.
Google Maps Integration (1 mark)	Integrate Google Maps API to visualize trip destinations and plan routes between activities.	Display a map with markers for each activity. Show optimal route, travel distance, and estimated time.
Any Additional Feature (1 mark each)	Any other creative features that improve usability or functionality.	Example: Suggest popular places, restaurants, or landmarks near an activity location.