OOP PROJECT

The OOP Project is an opportunity for bootcamp students to

- Begin to pull together a number of different things they've learned so far, including classes and polymorphism, user input, arrays and Lists, and exceptions;
- Work on a larger and more complicated case study than lab exercises as a prelude to the Final Project;
- Collaborate with teammates on a software project, experiencing the need for proper object-oriented design, documentation, and version control.
- Every student will participate in this project in an assigned group.
- The majority of class time will be committed to the project (although outside work, such as Soft Skill assignments, may still be necessary)
- There will be multiple check-ins during those days to make sure groups are on-track.
- The entire group will work together on one of the three projects. Take a little time to decide which one you want to tackle, but once you get started stick to that project—no turning back!

Possible projects:

- Point-of-Sale Terminal (a cash register/ordering terminal for someplace like a store, coffee shop, or fast-food restaurant)
- Library System
- Fitness Center

See the following pages for more information on each project. Please recognize that the descriptions are minimum versions; it's hoped each group will go beyond these requirements and incorporate features of interest to them.

Let your instructors and Teaching Assistant know which project your team has chosen and send them your GitHub repo link in Slack as soon as you have it.

POINT OF SALE TERMINAL

Write a cash register or self-service terminal for some kind of retail location. Obvious choices include a small store, a coffee shop, or a fast food restaurant. ● Your solution must include some kind of a product class with a name, category, description, and price for each item.

- 12 items minimum; stored in a the application.
- Present a menu to the user and let them choose an item (by number or letter).
    - o Allow the user to choose a quantity for the item ordered.
    - o Give the user a line total (item price * quantity).
- Either through the menu or a separate question, allow them to re-display the menu and to complete the purchase.
- Give the subtotal, sales tax, and grand total. (Remember rounding issues the Math library will be handy!)
- Ask for payment type—cash, credit, or check
- For cash, ask for amount tendered and provide change.
- For check, get the check number.
- For credit, get the credit card number, expiration, and CVV.
- At the end, display a receipt with all items ordered, subtotal, grand total, and appropriate payment info.
- Return to the original menu for a new order.

Optional enhancements:

- Store your list of products in a text file and then include an option to add to the product list, which then outputs to the product file.

LIBRARY TERMINAL

Write a console program which allows a user to search a library catalog and check out books.

- Your solution must include some kind of a book class with a title, author, status, and due date if checked out.
    - Status should be On Shelf or Checked Out (or other statuses you can imagine).

- 12 items minimum; All stored in the the application.

- Allow the user to:
    - Display the entire list of books. Format it nicely.
    - Search for a book by author.
    - Search for a book by title keyword.
    - Select a book from the list to check out.
        - If it's already checked out, let them know.
        - If not, check it out to them and set the due date to 2 weeks from today.
            - Return a book. (You can decide how that looks/what questions it asks.)

Optional enhancement:

- When the user quits, save the current library book list (including due dates and statuses) to the text file so the next time the program runs, it remembers.

FITNESS CENTER

Write a console application for a fitness center to help manage members and membership options. At a minimum, this program should include:

- A class to hold basic details about Members (this class should eventually have at least 2 child classes) and hold the following details at a minimum:
    - id, name
    - an abstract method void CheckIn(Club club)
- A minimum of two child classes that represent a Single Club Member and Multi-Club Members (these members can visit various locations using the same membership).

  The classes should have the following:

  - Single Club Members: a variable that assigns them to a club.
    The CheckIn method throws an exception if it's not their club.

  - Multi-Club Members: a variable that stores their membership points.
    The CheckIn method adds to their membership points.

- A Club class that holds basic details about each fitness club, including at minimum:
    - name
    - address
    - anything else you think might be useful

- Allow users to:
    - Add members (both kinds), remove members or display member information.
    - Check a particular member in at a particular club. (Call the CheckIn method).
    - Select a member and generate a bill of fees. Include membership points for Multi-Club Members.

- A provide a main class which takes input from the user:
    - Asks a user if they want to select a club
    - Added members should be given the option to select from at least 4 fitness center locations or have the option to be a multi-club member.
- Display a friendly error message if there is an exception. Don't let it crash the program.

Optional enhancements:

- Allow new members to receive discounts if they sign up during certain time periods, explore the DateTime library for help with date and time.

- Store clubs and members in text files.

OOP Project Setup Checklist

1. Pick a breakroom for the team to use for the project. You will use this breakout room throughout the duration of the project.
2. Review the project choices and make a team decision on which one you choose.
3. Let your instructor know which project you have chosen.
4. **ONE TEAM MEMBER** creates a project repository in GitHub:
    a. **Repository Name**: Should include the name of the project. e.g. *team1-fitness-center*
    b. **Access: public**
    c. **Git ignore template**: **VisualStudio** (it's near the bottom of the list)
    d. Create the repository
    e. Copy the URL and share it with your team members in the breakout room chat.
5. Once the project repository is created, **each team member** should clone it down to their machine:
    a. Open a Terminal or GitBash session.
    b. Navigate to your workspace directory (the one with your classwork and assessment repos)
    c. Issue the command **git status** (you should get a **message saying it is NOT a git repository**; **if it says anything else or you are not sure, contact your instructor before proceeding any further**)
    d. Copy the repo URL from the breakout room chat.
    e. Clone the repo down to your machine:

        git clone *paste-the-url-for-the-repo*

    f. Navigate to the project repo folder. If you forgot the name do an **ls** and you should see the name.
6. **ONE TEAM MEMBER** should:
    a. Open up Rider or Visual Studio
    b. Create a new Console Application solution in the project repo folder.
    c. **BE SURE YOU CREATE THE SOLUTION IN THE PROJECT REPO FOLDER!**
    d. **BE SURE YOUR PROJECT HAS A Program.cs FILE WITH A CLASS MAIN!**
    e. Save the project and exit the IDE.
    f. Go to your Terminal or GitBash session
    g. Navigate to your project folder (or verify you are in it by doing a **pwd**)
    h. Do a **git status** and verify your the solution project you created is listed
    i. Push the new solution project so you team mates can get the code:

        git add -A
        git commit -m'initial commit'
        git push origin main

    j. Do a **git status** and verify there is nothing left to push.

(Continued on next page)

OOP Project Setup Checklist *(continued)*

7. **ALL TEAM MEMBERS EXCEPT THE ONE THAT CREATED THE SOLUTION PROJECT** should:
    a. Navigate to your project folder (or verify you are in it by doing a **pwd**).
    b. Do a **git status** to verify there is nothing to push.
    c. Download the solution project from the project repo:

        git pull origin main

    d. Do an **ls** to verify the solution project has been downloaded to you machine

Please do not hesitate to contact your instructor or Teaching Assistant if you're unsure of any of this, or anything else, before you attempt it. It is much easier to avoid a problem by asking for help than trying to fix one.

Once you have your repos and branches set up, you should start thinking about the project.

As a team:

1. Analyze the problem statement/requirements identifying the major/big pieces /processes.
2. Break the major pieces down into smaller manageable pieces
3. Identify objects required by the process:
    a. Name/Purpose/Use
    b. Data required *(these become your data members)*
    c. Behaviors it should exhibit/Processing it should do *(these become your class methods)*
    d. Its relationship to other objects: *has-a is-a is-type-of*
4. Decide which classes or features you want to implement first.
5. Assign one class/feature to each team member.

It is not unusual for the analysis of the problem and designing the pieces of the solution to take longer than expected. **Resist the urge to just start coding.**

Team members should only be working on and making changes to files associated with their piece.

Avoid having more than one person making changes to any file in the project. This will usually lead to merge conflicts. Even adding or changing spacing in a file that someone is working on may cause a merge conflict. **Contact your instructor if you are having git problems or are unsure if you do.**

Remember you are a team of equals, respect each other's skills and abilities. If one of your team mates is struggling, look at it as a learning opportunity for YOU to help them. Don't just give them the answer, help them understand and come to the answer themselves.

Communication is the lifeblood of the team, be sure your teammates are always aware of your status, availability and skill level.

PRESENTATIONS GUIDELINES

At the end of the project, we will present our build to both staff and students. This doesn't need to be anything fancy, just a quick run through of your code.

Here's the rules for presentations each group should:

- Take 10 minutes maximum to present
- Show off the project running
- Show off the most important objects and methods in the code ● Don't go line by line, quick few sentence summaries are more than enough detail
- Share the presentation between group members roughly equally, everyone should speak
- Focus on the back-end, meaning transforming and manipulating data. We care about how the pizza is made, not how it looks. (EG: we check out a book update the due date in the book, and then update it in our File)
- Talk about the challenges you faced and how you worked through them as a group (Employers looove hearing about this exact kinda stuff)