Design Write-up
CPSC 231
Studebaker
Brooke England + Raneem Rahman

Since this assignment is heavily focused on your design decisions as an Object-Oriented Programmer, you will also be submitting a written portion of your final assignment that will make up a portion of your final project grade.

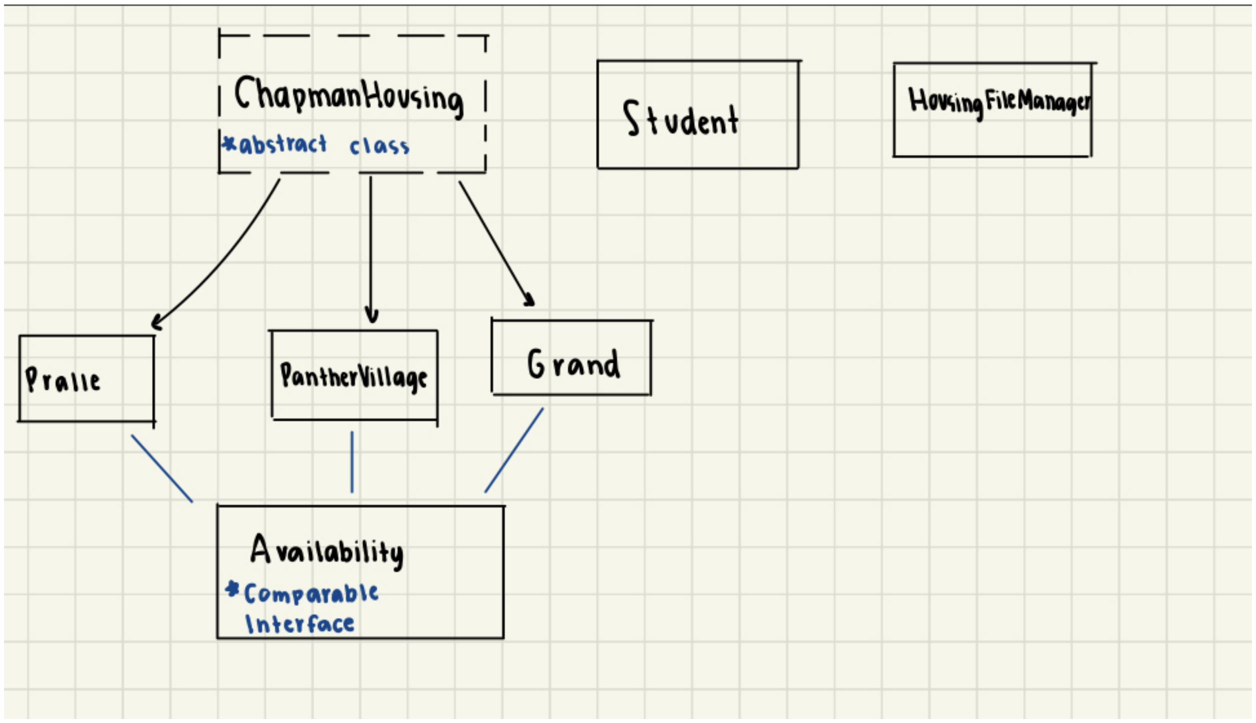
The Write-Up will be due one week before the finalized code for the final project. You should create a PDF including a detailed overview of how you've chosen to organize this project and the decisions involved while developing. Make sure to include:

- **A high level overview of your project and why you chose to build it**
  - The project creates a program allowing users to explore available dormitories at Chapman housing. Users can view amenities (laundry, parking, gym) roommate details, bathroom configurations, and dormitory locations. They can also check availability and be added or removed from a dorm room.
- **A description of each class and interface: their purpose, member variables, and methods**
  1. ChapmanHousing
     - Abstract class
     - Represents a generic dorm, captures common attributes shared by the different types of Chapman Housing. Creates a baseline for the properties and methods that are common for all kinds of Chapman Housing
     - Serves as the parent/base class for the different types of housing
     - **Member Variables:** roomsAvailable (int), numRoomates (int), currentOccupancy (int), sharedBathroom (boolean), price (double), shuttle (boolean), gym (boolean), parking (boolean), numLaundry (int), mealPlan (boolean), location (String "on campus" or "off campus"), housingType (String "freshman dorm" or "apartment"), buildingCapacity (int), occupants (list of students)
     - Abstract Method: calcOccupancy() method that calculates how full the building currently is. The implementation depends on which dorm/apartment it is.
     - Methods: addStudent(), removeStudent() adds and removes students from dorm/apartment buildings.
  2. Pralle
     - Inherits from ChapmanHousing
     - Represents Pralle dorm building
     - Member variables: occupants (a list of students), inherits member variables from ChapmanHousing

- Methods: Accessors and mutators, equals(), toString(), implements addStudent(), implements removeStudent(), implements calcOccupancy(), compareTo() implements comparable interface

3. Grand
    - Inherits from ChapmanHousing
    - Represents Grand apartments
    - Member variables: numBedrooms (int), inherits member variables from ChapmanHousing
    - Methods: Accessors, Mutators, equals(), toString(), implements addStudent(), implements removeStudent(), implements calcOccupancy(), compareTo() implements comparable interface
4. Panther Village
    - Inherits from ChapmanHousing
    - Represents an apartment at Panther Village
    - Member variables: numBedrooms (int), inherits member variables from ChapmanHousing
    - Methods: Accessors, Mutators, equals(), toString(), implements addStudent(), implements removeStudent(), implements calcOccupancy(), compareTo() implements comparable interface
5. Availability
    - Comparable interface
    - Tells the user whether there is space available in a dorm room
    - Methods: calcSpaceAvailable()
6. chapmanStudent
    - Represents a Chapman student
    - Member variables: studentID, name, currentResidence
    - Accessors, Mutators, equals(), toString()
7. HousingFileManager
    - Handles input and output operations for housing data
    - Methods: saveDataToFile()

- **A diagram (hand drawn or digital) of the relationships between your classes and interfaces**



- **Why you've chosen that particular architecture instead of an alternative approach**
  - We're using abstraction through the abstract class "ChapmanHousing"
  - By using this particular architecture, it will be easier to expand on the different dorm types and what they have to offer while still having the basic functioning interface
- **How you intend to incorporate File I/O into your project and what data will be loaded in and saved to a file, along with how that data will be structured**
  - Will be able to save data about students info to a file HousingFileManager class. Students will be organized by their info including name, student ID, and the dorm they are in.
- **Whether or not you intend to use any additional Java packages or APIs (some students want the challenge of learning to build a UI, play audio, fetch data from the internet, etc)**
  - We are not going to include any additional Java packages or APIs
- **How you intend to divide the work between the members of your group**
  - We are going to divide up the coding equally
  - We will each be responsible for implementing different classes, keeping up with coding standards, and regularly checking in with one another to make sure our code is compatible. Since there are 7 classes, we will each do 3 separate classes and work on one together. We will then combine our classes into a single functioning program.

- **Places you could see future developers expanding upon your project and adding additional features**
  - Future developers could definitely expand on our project and potentially use it for other schools so navigate their housing systems
  - They can maybe include search and filter options, or even real time updates for occupancy rates and availability
  - Developers can also develop the code and use additional Java packages or API's to make it more interactive for the user
  - Developers could expand our current code by adding more specific tasks that the user can perform such as viewing the individual rooms in the buildings, number of beds/desks/dressers, check in/checkout dates, and more information about the roommates such as age, major etc.