Our program utilizes various SOLID and GRASP design concepts to detail the relations between the classes, with 5 specific concepts of Single Responsibility Principle (SOLID), Open/Closed Principle (SOLID), Liskov Substitution Principle (SOLID), ISP, and \_\_\_\_\_\_

1. The program practices the Single Responsibility Principle (SRP), the first part of the SOLID principles, through the singular responsibility of each non-generic class of Admin and GeneralMember. By implementing a general overarching abstract class of Member along with an interface OnProject above that, we declare the general behaviors before we implement the specific member classes of GeneralMember and Admin. Specifically, the “OnProject” interface declares the joinProject and leaveProject behaviors of various members of the project. Therefore, when we implement the specific members of the Admin and GeneralMember class, each of these specific member classes have only a singular function/behavior that is unique to them aside from the inherited ones. For the Admin class, this behavior is givePermission(), while for the general member class, this behavior is requestPermission. By following the SRP, these specific member classes then only have this singular behavior along with the general behaviors of joinProject and leaveProject that they already inherit, utilizing only one singular responsibility for each specific class.
2. The program also uses the Open/Closed Principle, the second part of SOLID principles. The open/closed principle emphasizes the importance for a program to be open for extensions, but closed for modifications. A vital method to implement this practice is to use interfaces, since we can separate behaviors so that the core of the system cannot be altered. As stated above, our project program uses an OnProject interface that is extended by the members abstract class, which is again extended by the specific GeneralMember and Admin classes. This direct sequence of extensions using interfaces along with separation of tasks declares general behaviors, such as joinProject or leaveProject, in the interface of OnProject but defines them in the specific classes. This demonstrates the compliant use of the Open/Closed Principle to extend the classes freely but keep them closed for modification.
3. Creator (Project.java and Task.java): The Project class embodies the Creator GRASP principle through its direct responsibility for creating and managing task objects, demonstrated by methods such as addTask(Task newTask). This implementation aligns with the Creator principle as the Project class naturally contains and organizes tasks, making it the most suitable class to create instances of Task. This design choice simplifies the structure of the application by centralizing the creation of tasks within the context where they are directly relevant, thereby enhancing the code's readability and maintainability.
4. High Cohesion (Member.java): The Member class demonstrates High Cohesion by focusing exclusively on member-related functionalities. This class is responsible for handling attributes and behaviors pertinent to project members, such as their personal information, roles, and task assignments. By maintaining a clear and singular focus, the Member class avoids unrelated responsibilities, which enhances its clarity and effectiveness. High Cohesion in Member ensures that the class remains focused on a specific set of related tasks, promoting easier debugging, testing, and maintenance.
5. Liskov Substitution Principle (LSP) (Member.java, GeneralMember.java, Admin.java): The inheritance structure between Member, GeneralMember, and Admin upholds the LSP by ensuring that objects of GeneralMember and Admin can be used interchangeably in any context expecting a Member object without altering the desired behavior. This substitutability confirms that subclasses have been designed in accordance with their base class, maintaining behavioral consistency and reliability across the application.