Dokumentacja Projektu

Projekt z konsolowym interfejsem użytkownika, który ma za zadanie imitować system odpowiedzialny za zarządzanie projektami w firmie. Wszelkie dane wprowadzone przez użytkownika systemu są gromadzone w plikach tekstowych odtwarzających zachowania i funkcjonalność relacyjnych baz danych.

Spis Klas

"Bazowe" Klasy

Klasy odpowiedzialne za zapisywanie, kasowanie, modyfikowanie i wyszukiwanie danych oraz klasy do zarządzania interfejsem konsolowym.

`CompanyDB`

Klasa `CompanyDB` jest odpowiedzialna za przepływ danych oraz śledzi wszystkie zmiany wprowadzane przez użytkownika. Imituje bazę danych i gromadzi informacje o aktualnie trwających projektach oraz o członkach ich zespołów, a także o dodanych zadaniach i nadchodzących spotkaniach.

Zawiera również informacje o pracownikach. Są one przekazywane bezpośrednio przez firmę, zatem system nie jest w stanie w żaden sposób nimi manipulować.

Metody:

- `loadData()`: `void` - Wczytuje do systemu pracowników firmy i wywołuje funkcję `loadProjects()`.
- `clearFiles(Project project)`: `void` - Odpowiada za skasowanie wszystkich informacji związanych z projektem, który został usunięty z systemu. Wywołuje funkcję `clearFile()` na odpowiednich plikach tekstowych.
- `clearFile(String fileName, String projectIndex)`: `void` - Czyści podany plik tekstowy, kasując z niego wszystkie informacje dotyczące projektu, którego indeks został przekazany jako argument funkcji.
- `loadProjects()`: `void` - Wczytuje do systemu projekty, nad którymi aktualnie pracuje firma (czyli projekty, których indeksy widnieją w bazie).
- `loadTasks(Project project)`: `void` - Wczytuje do systemu aktualne informacje o zadaniach przypisanych do projektu, który został przekazany jako argument funkcji.
- `loadMembers(Project project)`: `void` - Wczytuje do systemu aktualne informacje o członkach zespołu pracujących nad projektem, który został przekazany jako argument funkcji.
- `loadSchedule(Project project)`: `void` - Wczytuje do systemu aktualne informacje o harmonogramie dołączonym do projektu, który został przekazany jako argument funkcji.
- `saveProjectsToFile()`: `void` - Zapisuje do odpowiedniego pliku tekstowego wszystkie modyfikacje, które zaszły w projektach podczas całej sesji działania programu.
- `saveTasksToFile(ArrayList<Task> tasks)`: `void` - Zapisuje do odpowiedniego pliku tekstowego wszystkie modyfikacje, które zaszły w zadaniach podczas całej sesji działania programu.
- `saveMembersToFile(ArrayList<TeamMember> members)`: `void` - Zapisuje do odpowiedniego pliku tekstowego wszystkie modyfikacje, które dotyczyły członków zespołu, a które zaszły podczas całej sesji działania programu.

- `saveScheduleToFile(ArrayList<Meeting> meetings)`: `void` - Zapisuje do odpowiedniego pliku tekstowego wszystkie modyfikacje, które zaszły w harmonogramie podczas całej sesji działania programu.

Main

Klasa `Main` oprócz tego, że posiada metodę uruchamiającą cały program, posiada także własne metody oraz atrybuty. Steruje pierwszymi wyborami użytkownika, który decyduje o tym, jaki projekt utworzyć, usunąć lub który zmodyfikować.

Atrybuty:

- `projectsCounter`: `int` - Zmienna odpowiedzialna za nadawanie projektom unikalnych indeksów.
- `companyWorkers`: `ArrayList<Worker>` - Lista zwykłych pracowników firmy.
- `companyProjectManagers`: `ArrayList<ProjectManager>` - Lista osób zarządzających projektami w firmie.
- `companyProjects`: `ArrayList<Project>` - Lista rozpoczętych projektów w firmie.

Metody:

- `getWorkerByIndex(int index)`: `Worker` - Zwraca pracownika o przekazanym indeksie.
- `getPMByIndex(int index)`: `ProjectManager` - Zwraca kierownika projektów o przekazanym indeksie.
- `getProjectByIndex(int index)`: `Project` - Zwraca projekt o przekazanym indeksie.
- `addProject(Scanner scanner)`: `void` - Umożliwia użytkownikowi dodanie nowego projektu do systemu.
- `removeProject(Scanner scanner)`: `void` - Usuwa z systemu projekt wskazany przez użytkownika. Wywołuje funkcję `clearFiles()` klasy `CompanyDB`.
- `manageProject(Scanner scanner)`: `void` - Po wybraniu przez użytkownika projektu do modyfikacji wywołuje funkcje `loadMembers()`, `loadSchedule()` oraz `loadTasks()` klasy `CompanyDB` z odpowiednim argumentem (projektem wybranym przez użytkownika), a następnie wywołuje funkcję `manageProject()` klasy `Managing`.

Managing

Klasa `Managing` wyświetla użytkownikowi listę widoków, spośród których użytkownik może wybrać interesującą go opcję. Każdy z widoków wyświetli udostępnianą przez niego listę możliwych modyfikacji. Na podstawie wyboru użytkownika wywoływana jest odpowiednia funkcja klasy `Views`.

Metody:

- `manageProject(Scanner scanner)`: `void` - Wyświetla użytkownikowi listę widoków, a wybranie któregokolwiek z nich (przez użytkownika) wywoła odpowiednią funkcję klasy `Views`. W przypadku wyboru ostatniej opcji, a więc cofnięcia się do widoku klasy `Main`, wywołuje funkcje `saveTasksToFile()`, `saveMembersToFile()` oraz `saveScheduleToFile()`, aby wszelkie wprowadzone przez użytkownika zmiany zostały zapisane w plikach tekstowych.

Views

Klasa `Views` jest zbiorem wszystkich działań, które może podjąć użytkownik, zarządzając konkretnym projektem. Po wyborze odpowiedniego widoku, użytkownikowi ukazuje się lista możliwych operacji dostępnych w danym widoku.

Metody:

- `taskManagerView(Scanner scanner, Project project)`: `void` - Wyświetla użytkownikowi listę dostępnych działań dotyczących *zadań* przypisanych do projektu. Każdy z wyborów użytkownika wywołuje inną funkcję klasy `TaskManager`.
- `teamView(Scanner scanner, Project project)`: `void` - Wyświetla użytkownikowi listę dostępnych działań dotyczących *zespołu* przypisanego do projektu. Każdy z wyborów użytkownika wywołuje inną funkcję klasy `Team`.
- `projectView(Scanner scanner, Project project)`: `void` - Wyświetla użytkownikowi listę dostępnych działań dotyczących danego *projektu*. Każdy z wyborów użytkownika wywołuje inną funkcję klasy `Project`.
- `teamMemberView(Scanner scanner, Project project)`: `void` - Po udanej weryfikacji (sprawdzeniu, czy użytkownik należy do zespołu pracującego nad projektem), wyświetla użytkownikowi listę dostępnych działań dla tego *pracownika*. Każdy z wyborów użytkownika wywołuje inną funkcję klasy `TeamMember`.
- `projectManagerView(Scanner scanner, Project project)`: `void` - Po udanej weryfikacji (sprawdzeniu, czy użytkownik jest odpowiedzialny za kierowanie projektem), wyświetla użytkownikowi listę dostępnych działań dla tego *pracownika*. Każdy z wyborów użytkownika wywołuje inną funkcję klasy `ProjectManager`.
- `scheduleView(Scanner scanner, Project project)`: `void` - Wyświetla użytkownikowi listę dostępnych działań dotyczących *harmonogramu* przypisanego do projektu. Każdy z wyborów użytkownika wywołuje inną funkcję klasy `Schedule`.

Pozostałe Klasy

Klasy obiektów, które razem umożliwiają efektywne zarządzanie projektem w firmie.

`Worker`

Klasa `Worker` reprezentuje pracownika firmy. Przechowuje podstawowe informacje takie jak unikalny indeks, imię, nazwisko oraz adres e-mail.

Program nie daje możliwości tworzenia obiektów tej klasy. Pracownicy firmy są przekazywani do systemu za pomocą metody `loadData()` klasy `CompanyDB`.

Konstruktor:

```
public Worker(int index, String firstName, String lastName, String email,
              boolean addToCompanyWorkers)
```

Atrybuty:

- `index`: `int` - Unikalny indeks pracownika.
- `firstName`: `String` - Imię pracownika.
- `lastName`: `String` - Nazwisko pracownika.
- `email`: `String` - Adres e-mail pracownika.

Metody:

- `displayInfo()`: `void` - Wyświetla informacje o pracowniku.
- `getIndex()`: `int` - Zwraca indeks pracownika.
- `getFirstName()`: `String` - Zwraca imię pracownika.
- `getLastName()`: `String` - Zwraca nazwisko pracownika.
- `getEmail()`: `String` - Zwraca email pracownika.

`ProjectManager`

Klasa `ProjectManager` reprezentuje kierownika projektu. Klasa dziedziczy po klasie `Worker`, jednak przechowuje dodatkowo informacje o projektach, w które zaangażowany jest kierownik projektu.

Obiektów tej klasy, podobnie jak obiektów klasy `Worker`, nie da się dodać w programie. Dodatkowo indeksy kierowników projektów mogą pokrywać się z indeksami pracowników, ponieważ są to dwie rozróżnialne klasy obiektów.

Konstruktor:

```
public ProjectManager(int index, String firstName, String lastName, String email)
```

Atrybuty:

- *Odziedziczone po klasie* `Worker`
- `projects`: `ArrayList<Project>` - Lista projektów, którymi aktualnie zarządza kierownik projektów.

Metody:

- *Odziedziczone po klasie* `Worker`
- `showProjects()`: `void` - Wyświetla listę projektów, którymi aktualnie zarządza kierownik projektów.
- `addProject(Project project)`: `void` - Dodaje nowy projekt do listy projektów danego kierownika projektów.

TeamMember

Klasa `TeamMember` reprezentuje członka zespołu pracującego nad projektem. Klasa dziedziczy po klasie `Worker`, jednak przechowuje dodatkowe informacje takie jak unikalny indeks projektu, nad którym pracuje dany członek zespołu oraz nadany mu stopień uprawnień.

Obiekty tej klasy powstają automatycznie podczas wywołania metody `addNewMember()` na obiekcie klasy `Team`.

Konstruktor:

```
public TeamMember(int index, int projectIndex, String firstName, String lastName,
                  String email, int permissionStatus)
```

Atrybuty:

- *Odziedziczone po klasie* `Worker`
- `projectIndex`: `int` - Unikalny indeks projektu, nad którym pracuje członek zespołu.
- `permissionStatus`: `int` - Stopień uprawnień, jaki posiada dany członek zespołu. Określa on, które zadania mogą zostać mu przydzielone.

Metody:

- *Odziedziczone po klasie* `Worker`
- `getPermissionStatus()`: `int` - Zwraca stopień uprawnień członka zespołu.
- `showTasks(Project project)`: `void` - Wyświetla listę zadań przydzielonych pracownikowi w ramach danego projektu.
- `addToCommonFile(Scanner scanner, Project project)`: `void` - Umożliwia pracownikowi modyfikację wspólnego pliku, poprzez dodanie idei, którą chciałby się podzielić z zespołem.
- `parseToString()`: `String` - Zbiera informacje, po których można zidentyfikować członka zespołu i tworzy z nich napis typu `String`. Metoda potrzebna do zapisu danych do plików tekstowych.
- `loadFromFile(String line)`: `TeamMember` - Odtwarza członka zespołu na podstawie informacji zawartych w przekazanym napisie typu `String`. Metoda potrzebna do pobierania i odczytywania danych z plików tekstowych.

Team

Klasa `Team` reprezentuje zespół, który pracuje nad danym projektem. Klasa przechowuje informacje takie jak nazwa zespołu oraz lista pracowników, którzy do niego należą.

Obiekt tej klasy powstaje automatycznie podczas inicjalizacji obiektu klasy `Project`, stąd jest przypisany do konkretnego projektu.

Atrybuty:

- `nickname`: `String` - Nazwa zespołu postaci "ProjectNo__%d__", gdzie %d jest unikalnym indeksem projektu, w ramach którego stworzono zespół.
- `members`: `ArrayList<TeamMember>` - Lista członków zespołu.

Metody:

- `initializeMembersArray()`: `void` - Tworzy w systemie listę, która będzie przechowywać informacje o członkach należących do zespołu.
- `setNickname(String nickname)`: `void` - Ustanawia nazwę zespołu.
- `getNickname()`: `String` - Zwraca nazwę zespołu.
- `getMembers()`: `ArrayList<TeamMember>` - Zwraca listę członków zespołu.
- `addMember(TeamMember teamMember)`: `void` - Dodaje osobę do listy członków zespołu. Metoda potrzebna do przekazania systemowi aktualnych informacji zapisanych w plikach tekstowych.
- `displayInfo()`: `void` - Wyświetla informacje o zespole.
- `addNewMember(Scanner scanner, Project project)`: `void` - Dodaje wskazanego przez użytkownika pracownika, wraz z jego uprawnieniami, do zespołu.
- `removeMember(Scanner scanner)`: `void` - Usuwa wskazanego przez użytkownika pracownika z zespołu.
- `getMemberByIndex(int index)`: `TeamMember` - Zwraca członka zespołu na podstawie indeksu.

Meeting

Klasa `Meeting` reprezentuje pojedyncze spotkanie. Przechowuje informacje takie jak indeks projektu, którego dotyczy, nazwa, miejsce, czas oraz przewidywana długość trwania spotkania.

Konstruktor:

```
public Meeting(int projectIndex, String title, String place, LocalDateTime time,
               double duration)
```

Atrybuty:

- `projectIndex`: `int` - Indeks projektu, którego dotyczy spotkanie.
- `title`: `String` - Nazwa spotkania.
- `place`: `String` - Miejsce spotkania.
- `time`: `LocalDateTime` - Czas (data i godzina) spotkania.
- `duration`: `double` - Przewidywany czas spotkania podawany w godzinach.

Metody:

- `displayInfo()`: `void` - Wyświetla informacje o spotkaniu.
- `parseToString()`: `String` - Zbiera informacje, po których można zidentyfikować spotkanie i tworzy z nich napis typu `String`. Metoda potrzebna do zapisu danych do plików tekstowych.
- `loadFromFile(String line)`: `Meeting` - Odtwarza spotkanie na podstawie informacji zawartych w przekazanym napisie typu `String`. Metoda potrzebna do pobierania i odczytywania danych z plików tekstowych.

Schedule

Klasa `Schedule` reprezentuje harmonogram przypisany do projektu. Przechowuje informacje o nadchodzących spotkaniach.

Obiekt tej klasy powstaje automatycznie podczas inicjalizacji obiektu klasy `Project`, stąd jest przypisany do konkretnego projektu.

Atrybuty:

- `meetings`: `ArrayList<Meeting>` - Lista nadchodzących spotkań.

Metody:

- `initializeMeetingsArray()`: `void` - Tworzy w systemie listę, która będzie przechowywać informacje o spotkaniach.
- `getMeetings()`: `ArrayList<Meeting>` - Zwraca listę spotkań.
- `addMeeting(Meeting meeting)`: `void` - Dodaje spotkanie do harmonogramu. Metoda potrzebna do przekazania systemowi aktualnych informacji zapisanych w plikach tekstowych.
- `showMeetings()`: `void` - Wyświetla spotkania zapisane w harmonogramie.
- `addNewMeeting(Scanner scanner, Project project)`: `void` - Dodaje do harmonogramu spotkanie przekazane przez użytkownika.

Task

Klasa `Task` reprezentuje pojedyncze zadanie do wykonania w ramach projektu. Przechowuje informacje takie jak unikalny indeks zadania, indeks projeku, do którego jest przypisane, indeks pracownika, który jest za nie odpowiedzialny, nazwa oraz opis zadania, wymagany poziom uprawnień, termin oraz szacowany koszt wykonania i status zadania.

Obiektami tej klasy zarządza obiekt klasy `TaskManager`.

Konstruktor:

```
public Task(int memberIndex, String name, String description,
            int requiredPermissionStatus, LocalDate deadline,
            double estimatedCost)
```

Atrybuty:

- `index`: `int` - Unikalny indeks zadania.
- `projectIndex`: `int` - Indeks projektu, w ramach którego powstało zadanie.
- `memberIndex`: `int` - Indeks pracownika odpowiedzialnego za dane zadanie.
- `name`: `String` - Nazwa zadania.
- `description`: `String` - Opis zadania.
- `requiredPermissionStatus`: `int` - Wymagany poziom uprawnień, który musi mieć członek zespołu, aby mógł podjąć się danego zadania.
- `deadline`: `LocalDate` - Termin wykonania zadania.
- `estimatedCost`: `double` - Szacowany koszt wykonania zadania.
- `isCompleted`: `boolean` - Informacja o tym, czy zadanie zostało wykonane.

Metody:

- `getIndex()`: `int` - Zwraca indeks zadania.
- `setIndex(int index)`: `void` - Ustanawia indeks zadania.
- `getProjectIndex()`: `int` - Zwraca indeks projektu.
- `setProjectIndex(int index)`: `void` - Ustanawia indeks projektu.
- `getMemberIndex()`: `int` - Zwraca indeks pracownika.
- `setMemberIndex(int newMemberIndex)`: `void` - Ustanawia indeks pracownika.
- `getName()`: `String` - Zwraca nazwę zadania.
- `getDescription()`: `String` - Zwraca opis zadania.
- `getRequiredPermissionStatus()`: `int` - Zwraca poziom uprawnień wymagany do podjęcia się danego zadania.
- `setNewDeadline(LocalDate deadline)`: `void` - Ustanawia nowy termin na wykonanie zadania.
- `setNewEstimatedCost(double newEC)`: `void` - Ustanawia nowy szacowany koszt wykonania zadania.
- `getEstimatedCost()`: `double` - Zwraca szacowany koszt wykonania zadania.
- `updateStatus()`: `void` - Zmienia status zadania na *wykonane*.
- `getStatus()`: `boolean` - Zwraca aktualny status zadania.
- `displayInfo()`: `void` - Wyświetla informacje o zadaniu.
- `parseToString()`: `String` - Zbiera informacje, po których można zidentyfikować zadanie i tworzy z nich napis typu `String`. Metoda potrzebna do zapisu danych do plików tekstowych.
- `loadFromFile(String line)`: `Task` - Odtwarza zadanie na podstawie informacji zawartych w przekazanym napisie typu `String`. Metoda potrzebna do pobierania i odczytywania danych z plików tekstowych.

TaskManager

Klasa `TaskManager` reprezentuje menedżera do spraw zadań. Przechowuje informacje takie jak lista zadań pozostałych do wykonania, lista zadań już wykonanych oraz licznik zadań.

Obiekt tej klasy powstaje automatycznie podczas inicjalizacji obiektu klasy `Project`, stąd jest przypisany do konkretnego projektu.

Atrybuty:

- `tasks`: `ArrayList<Task>` - Lista zadań pozostałych do wykonania.

- `completedTasks`: `ArrayList<Task>` - Lista wykonanych zadań.
- `tasksCounter`: `int` - Licznik zadań, nadaje nowym zadaniom ich unikalne indeksy.

Metody:

- `initializeTasksArray()`: `void` - Tworzy w systemie listę, która będzie przechowywać informacje o zadaniach.
- `addToTasks(Task task)`: `void` - Dodaje zadanie do listy. Metoda potrzebna do przekazania systemowi aktualnych informacji zapisanych w plikach tekstowych.
- `addToCompletedTasks(Task task)`: `void` - Dodaje zadanie do listy zadań wykonanych. Metoda potrzebna do przekazania systemowi aktualnych informacji zapisanych w plikach tekstowych.
- `getTasks()`: `ArrayList<Task>` - Zwraca listę zadań do wykonania.
- `getCompletedTasks()`: `ArrayList<Task>` - Zwraca listę wykonanych zadań.
- `showTasks()`: `void` - Wyświetla listę zadań do wykonania.
- `setAsCompleted(Scanner scanner, Project project)`: `void` - Zmienia status wskazanego przez użytkownika zadania na *wykonane*. Usuwa je z listy zadań do wykonania oraz przenosi je do listy wykonanych zadań. Aktualizuje budżet projektu, zmniejszając go o koszt wykonania zadania.
- `reassignTask(Scanner scanner, Project project)`: `void` - Przepisuje wskazane przez użytkownika zadanie innemu członkowi zespołu, również wskazanemu przez użytkownika.
- `changeTaskDeadline(Scanner scanner, Project project)`: `void` - Zmienia termin wykonania zadania na ten wskazany przez użytkownika.
- `updateTaskEstimatedCost(Scanner scanner, Project project)`: `void` - Zmienia szacowany koszt wykonania zadania na ten wskazany przez użytkownika.
- `addTask(Scanner scanner, Project project)`: `void` - Dodaje nowe, podane przez użytkownika zadanie do listy zadań do wykonania.
- `generateReport()`: `void` - Generuje raport na podstawie wykonanych oraz niewykonanych zadań. Wyświetla, jaka część projektu (procentowo) została zrealizowana.

`Project`

Klasa `Project` reprezentuje projekt, który ma zostać wykonany. Przechowuje informacje takie jak indeks projektu, nazwa oraz opis, termin wykonania, kierownik projektu, zespół pracujący nad projektem, menedżer do spraw zadań, harmonogram podpięty pod projekt, sponsor oraz przeznaczony budżet.

Podczas inicjalizacji obiektu tej klasy zostaje mu przypisany obiekt klasy `Team`, obiekt klasy `Schedule` oraz obiekt klasy `TaskManager`.

Konstruktor:

```
public Project(String name, String description, LocalDate deadline,
               ProjectManager projectManager, String sponsor, double budget)
```

Atrybuty:

- `index`: `int` - Indeks projektu.
- `name`: `String` - Nazwa projektu.
- `description`: `String` - Szczegółowy opis projektu.

- `deadline`: `LocalDate` - Termin wykonania projektu.
- `projectManager`: `ProjectManager` - Kierownik przydzielony do projektu.
- `team`: `Team` - Zespół przydzielony do prac nad projektem.
- `taskManager`: `TaskManager` - Menedżer zadań przydzielony do zarządzania zadaniami.
- `schedule`: `Schedule` - Harmonogram przydzielony do projektu.
- `sponsor`: `String` - Sponsor projektu.
- `budget`: `double` - Budżet przeznaczony na projekt.

Metody:

- `getIndex()`: `int` - Zwraca indeks projektu.
- `getProjectManager()`: `ProjectManager` - Zwraca kierownika projektu.
- `getTeam()`: `Team` - Zwraca zespół pracujący nad projektem.
- `getTaskManager()`: `TaskManager` - Zwraca menedżera zadań.
- `getSchedule()`: `Schedule` - Zwraca harmonogram spotkań projektu.
- `setBudget(double budget)`: `void` - Ustanawia budżet przeznaczony na projekt.
- `getBudget()`: `double` - Zwraca budżet przeznaczony na projekt.
- `getName()`: `String` - Zwraca nazwę projektu.
- `displayInfo()`: `void` - Wyświetla podstawowe informacje o projekcie.
- `updateBudget(Scanner scanner)`: `void` - Zmienia budżet przeznaczony na projekt na kwotę wskazaną przez użytkownika.
- `checkBudget()`: `void` - Monitoruje budżet. Wyświetla wykorzystaną kwotę, kwotę potrzebną do zrealizowania jeszcze niewykonanych zadań oraz pozostały budżet lub informację o jego przekroczeniu.
- `timeLeft()`: `void` - Wyświetla, ile czasu (lat, miesięcy i dni) pozostało do zrealizowania projektu.
- `parseToString()`: `String` - Zbiera informacje, po których można zidentyfikować projekt i tworzy z nich napis typu `String`. Metoda potrzebna do zapisu danych do plików tekstowych.
- `loadFromFile(String line)`: `Project` - Odtwarza projekt na podstawie informacji zawartych w przekazanym napisie typu `String`. Metoda potrzebna do pobierania i odczytywania danych z plików tekstowych.