Design Patterns

CS 121, second year students

Assignment 2

September 19, 2022. Due to October 10.

Overview

In this assignment you have to add the new classes to the schema described in *Assignment 1*. Also, you should refactor their structure using the abstract classes and special class containers like dataclasses. Basic knowledge in object oriented design is required. Current class structure is recommended, but you may slightly change them keeping the main logic.

Description

1. Implement container class **PersonalInfo** using ***@dataclass*** decorator in Python 3 (or standard Java/C#/C++ class style)[1,2,3]. Also, implement @property decorator named name (see ref. 4, 5) which splits name (e.g., "Dude_name", "Dude_surname") into **first_name** and **second_name** and sets the corresponding attributes. *Hint: Python method some_string.split() can be used.*

PersonalInfo
+id: int +first_name: str +second_name: str +address: str +phone_number: str +email: str +position: int +rank: str +salary: float

2. Implement abstract base class ***Employee***. Note that the **personal_info** attribute is kinda private :) (use ***_personal_info field*** in Python 3), so add here property decorator (in setter property please make sure that you are assigning an object of **PersonalInfo**[4,5], use ***isinstance(object1, object2)***[6]**.** There are two methods which should be *abstract*. Also, if possible move the duplicate methods from **Developer**, **ProjectManager**, **QualityAssurance** to ***Employee***.

Employee
-personal_info : PersonalInfo
abstract methods *+calculate_salary() -> None* *+ask_sick_leave(project_manager: ProjectManager) -> bool*

3. Modify the **Developer** class so that it is inherited from ***Employee.*** Remove methods **assign**() and **unassign**() from the **Developer.**

Developer(*Employee*)
... +calculate_salary() -> None +ask_sick_leave(project_manager: ProjectManager) -> bool

4. Implement **AssignManagement** which will take care about assignment logic in Developer->Project and Project->Developer. Provide arbitrary assignment mechanics.

AssignManagement
+employee: Employee +project: Project ...
+assign() -> None +unassign() -> None

5. Implement **Task** (*replace **task** dictionary from assignment 1*) class and provide the association/aggregation with **Project** class. Remember that **Project** consists of many tasks. Each task is assigned to a specific developer via **Assignment**'s. Choose your own implementation for provided methods.

Task
+id: int +title: str +deadline: datetime +items: List[str] +status: Any *# is_done or in_progress* +related_project: str *# project title*
+implement_item(item_name: str) -> str +add_comment(comment: str) -> None

6. Use **Task** class with **Assignment** class.

Assignment
+received_tasks : dict[Task]
+get_tasks_to_date(date: datetime) -> List[Task]

7. Add to ***task_list*** attribute in **Project** class list of tasks id. Use the **AssignManagement** class for adding and removing employers. Methods **add_developer**() and **remove_developer**() must be replaced by **add_employee**() and **remove_employee**(). These methods use **AssignManagement** instances.

Project
... +task_list: list[int] # task_id's
... +get_specific_employees(employee_type) -> List[Employee]

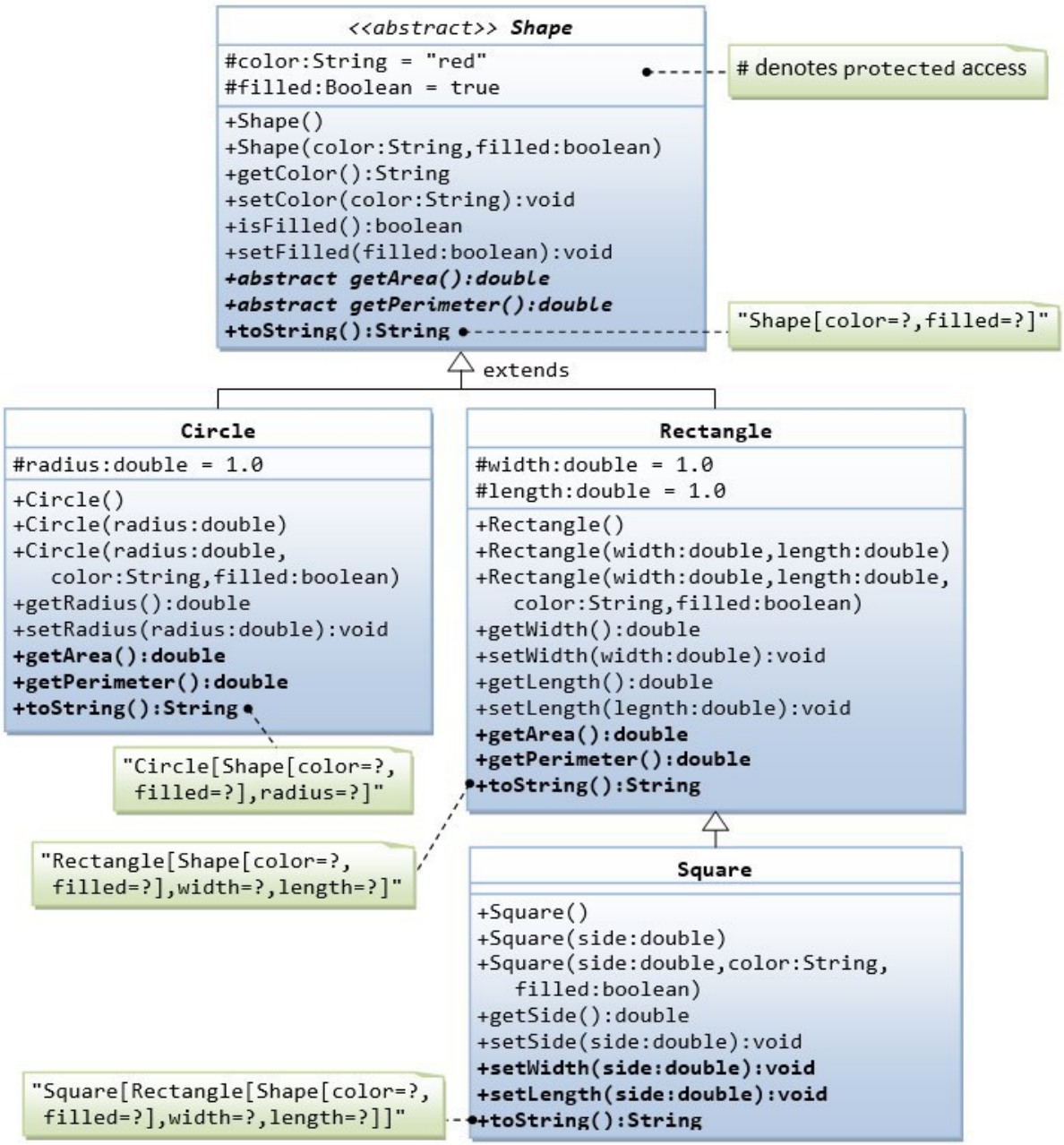
8. Modify **QualityAssurance, ProjectManager** classes so that they are inherited from ***Employee.***

QualityAssurance(*Employee*)
... +calculate_salary() -> None +ask_sick_leave(project_manager: ProjectManager) -> bool +add_ticket() -> None *# we will fill this method then, leave it blank.*

ProjectManager(*Employee*)
... +employee_requests: Any *# use this attribute to communicate with ask_sick_leave()*
+calculate_salary() -> None +discuss_progress(engineer: Employee)-> None *# we will fill this method then, leave it blank.*

Remark: you are free to add and modify proposed classes, but the main idea of the class communications and relations must be preserved.

- *Extra task to improve skills. Implement the following UML. Use 'pythonic' style for this implementation. Also, add unit tests[7,8]. Concrete shape can be defined by coordinates. Practice makes perfect.*



References:

1. https://www.geeksforgeeks.org/difference-between-dataclass-vs-namedtuple-vs-object-in-python/.
2. https://stackoverflow.com/questions/3357581/using-a-class-as-a-data-container
3. https://linuxhint.com/dataclasses-python/
4. https://www.programiz.com/python-programming/property
5. https://stackoverflow.com/questions/51079503/dataclasses-and-property-decorator
6. https://www.programiz.com/python-programming/methods/built-in/isinstance
7. https://www.digitalocean.com/community/tutorials/how-to-use-unittest-to-write-a-test-case-for-a-function-in-python
8. https://realpython.com/python-testing/
9. *Robert Martin*. Clean Code: A Handbook of Agile Software Craftsmanship.
10. *Mark Lutz*. Python, 5 ed.
11. *Dan Bader*. Clean Python.