**ZAMBIA UNIVERSITY COLLEGE OF TECHNOLOGY**

### logo-white

**SOFTWARE ENGINERING**

**SOFTWARE DESIGN 2210**

**COURSE CODE: BSE 2210**

**LECTURER NAME: MR CHIKWANDA**

**ASSIGNMENT: 2**

**GIVEN DATE: 21ST AUGUST 2024**

**DUE DATE:27TH AUGUST 2024**

STUDENT MANAGEMENT SYSTEM

**FIRSTNAME** **LASTNAME** **STUDENT NUMBER**

### CHOLA MAFUTA 2300205

### ANALYSIS

#### SOLID Principles

* Single Responsibility Principle (SRP):

**Violation**: The Student class handles both student data and updating student information. The Student Management System class handles both business logic and database operations.

**Example**: Student.update\_student method and Student Management System.update\_student\_info method.

* Open-Closed Principle (OCP):

**Violation:** The Student Management System class is not easily extendable without modifying existing code.

**Example**: Adding new functionalities like searching for a student would require modifying the StudentManagementSystem class.

* Liskov Substitution Principle (LSP):

**Violation**: Not directly violated, but ensuring sub classes can replace their parent classes without affecting the program is not demonstrated.

**Example**: No inheritance is used in the current code.

* Interface Segregation Principle (ISP):

**Violation**: Not directly violated, but the current design does not use interfaces or segregate them.

**Example:** All functionalities are lumped together in the Student Management System class.

* Dependency Inversion Principle (DIP):

**Violation**: High-level modules depend on low-level modules directly.

**Example**: Student Management System directly depends on Student Database.

#### DRY (Don’t Repeat Yourself)

**Meaning**: Avoid duplicating code or logic. Ensure that every piece of knowledge or logic is represented in a single, unambiguous place within the system.

**Violation:** Repeated logic for updating student information.

**Example:** Student.update\_student and Student Management System.update\_student\_info both handle updating student details.

#### KISS (Keep It Simple, Stupid)

**Meaning**: Keep the design and implementation of your code as simple as possible. Avoid overcomplicating things.

**Violation:** The design is more complex than necessary.

**Example**: The Student Management System class handles multiple responsibilities, making it harder to maintain.

#### YAGNI (You Ain’t Gonna Need It)

**Meaning**: Do not add functionality until it is necessary. Avoid implementing features that you think you might need in the future but are not required at the moment.

**Violation:** The current code does not have unnecessary features, but it could be simplified.

**Example**: The Student class has an update\_student method that could be simplified.

### Documentation

#### Key Sections Explained

* Student Class: Handles student data and updating student information.
* Student Database Class: Manages the collection of students, including adding, removing, and finding students.
* StudentManagementSystem Class: Provides high-level operations for managing students, delegating data operations to Student Database.
* Menu Function: Implements a simple text-based menu for user interaction.

NOT PART OF THE DOCUMENTS

### Documentation

**2.Run the System**:

python student\_management\_system.py

**3.Follow the Menu**:

* 1. The system will display a menu with options to add, delete, update, and view students.
  2. Enter the corresponding number to perform an action.

## Code Structure

### student\_management\_system.py

#### Classes

* **Student**:
  + Handles student data and updating student information.
  + Methods:
    - \_\_init\_\_(self, id, first\_name, last\_name, age, major): Initializes a new student.
    - update(self, first\_name=None, last\_name=None, age=None, major=None): Updates student information.
    - display(self): Displays student information.
* **StudentDatabase**:
  + Manages the collection of students.
  + Methods:
    - \_\_init\_\_(self): Initializes an empty student list.
    - add(self, student): Adds a student to the database.
    - remove(self, student\_id): Removes a student by ID.
    - find(self, student\_id): Finds a student by ID.
    - display\_all(self): Displays all students.
* **StudentManagementSystem**:
  + Provides high-level operations for managing students.
  + Methods:
    - \_\_init\_\_(self, database): Initializes with a database.
    - add\_student(self, id, first\_name, last\_name, age, major): Adds a new student.
    - delete\_student(self, student\_id): Deletes a student by ID.
    - update\_student(self, student\_id, first\_name=None, last\_name=None, age=None, major=None): Updates student information.
    - show\_all\_students(self): Displays all students.

#### Menu Function

* **menu()**:
  + Implements a simple text-based menu for user interaction.
  + Options to add, delete, update, and view students.

## Changes Made

* **Refactored Code**:
  + Followed SOLID principles to improve maintainability and scalability.
  + Eliminated redundancy by centralizing update logic.
  + Simplified design to make the code easier to understand and maintain.
  + Avoided unnecessary features to keep the code lean and efficient.

## Example Usage

system = StudentManagementSystem(StudentDatabase())

system.add\_student(1, "John Doe", 20, "Computer Science")

system.add\_student(2, "Jane Smith", 22, "Mathematics")

system.show\_all\_students()

system.update\_student(1, first\_name="Johnathan")

system.show\_all\_students()

system.delete\_student(2)

system.show\_all\_students()

## License

This project is licensed under the MIT License.

### Comments in Code

```python

class Student:

def \_\_init\_\_(self, id, first\_name, last\_name, age, major):

# Initialize a new student with ID, first name, last name, age, and major

self.id = id

self.first\_name = first\_name

self.last\_name = last\_name

self.age = age

self.major = major

def update(self, first\_name=None, last\_name=None, age=None, major=None):

# Update student information if new values are provided

if first\_name:

self.first\_name = first\_name

if last\_name:

self.last\_name = last\_name

if age:

self.age = age

if major:

self.major = major

def display(self):

# Display student information

print(f"\033[1mSTUDENT ID: {self.id}, First Name: {self.first\_name}, Last Name: {self.last\_name}, Age: {self.age}, Major: {self.major}\033[0m")

class StudentDatabase:

def \_\_init\_\_(self):

# Initialize an empty list to store students

self.students = []

def add(self, student):

# Add a student to the database

self.students.append(student)

def remove(self, student\_id):

# Remove a student by ID

self.students = [student for student in self.students if student.id != student\_id]

def find(self, student\_id):

# Find a student by ID

for student in self.students:

if student.id == student\_id:

return student

return None

def display\_all(self):

# Display all students in the database

if not self.students:

print("\033[1mNo students in the database.\033[0m")

else:

for student in self.students:

student.display()

class StudentManagementSystem:

def \_\_init\_\_(self, database):

# Initialize with a database

self.database = database

def add\_student(self, id, first\_name, last\_name, age, major):

# Add a new student to the system

student = Student(id, first\_name, last\_name, age, major)

self.database.add(student)

def delete\_student(self, student\_id):

# Delete a student by ID

if not self.database.students:

print("\033[1mCannot delete. The database is empty.\033[0m")

else:

self.database.remove(student\_id)

def update\_student(self, student\_id, first\_name=None, last\_name=None, age=None, major=None):

# Update student information

if not self.database.students:

print("\033[1mCannot update. The database is empty.\033[0m")

else:

student = self.database.find(student\_id)

if student:

student.update(first\_name, last\_name, age, major)

else:

print("\033[1mStudent not found.\033[0m")

def show\_all\_students(self):

# Display all students in the system

self.database.display\_all()

def menu():

# Create a new database and management system

database = StudentDatabase()

system = StudentManagementSystem(database)

while True:

# Display menu options

print("\n1. Add Student")

print("2. Delete Student")

print("3. Update Student")

print("4. Show All Students")

print("5. Exit")

# Get user choice

choice = input("Enter your choice: ")

if choice == '1':

REFERENCES

Chris, K. (2023) SOLID design principles in software development. <https://www.freecodecamp.org/news/solid-design-principles-in-software-development/.>