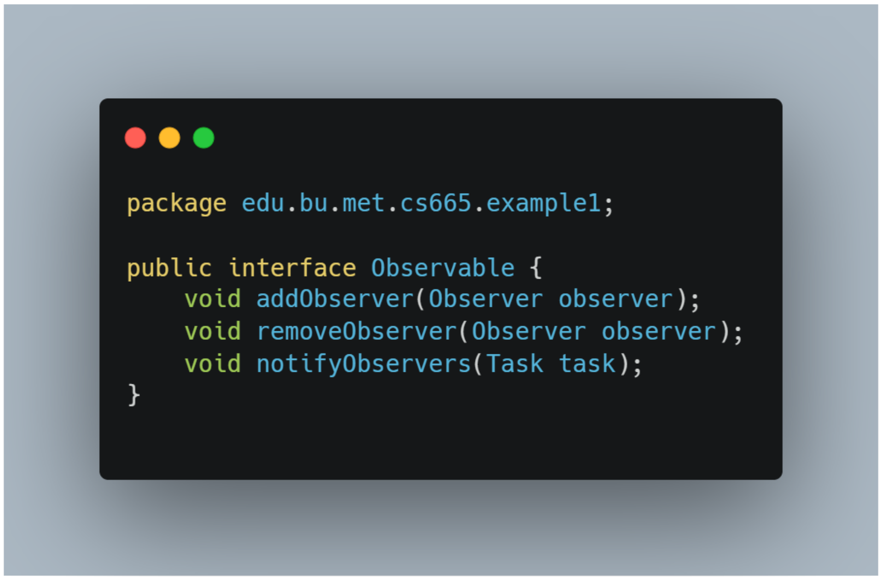
Assignment 3

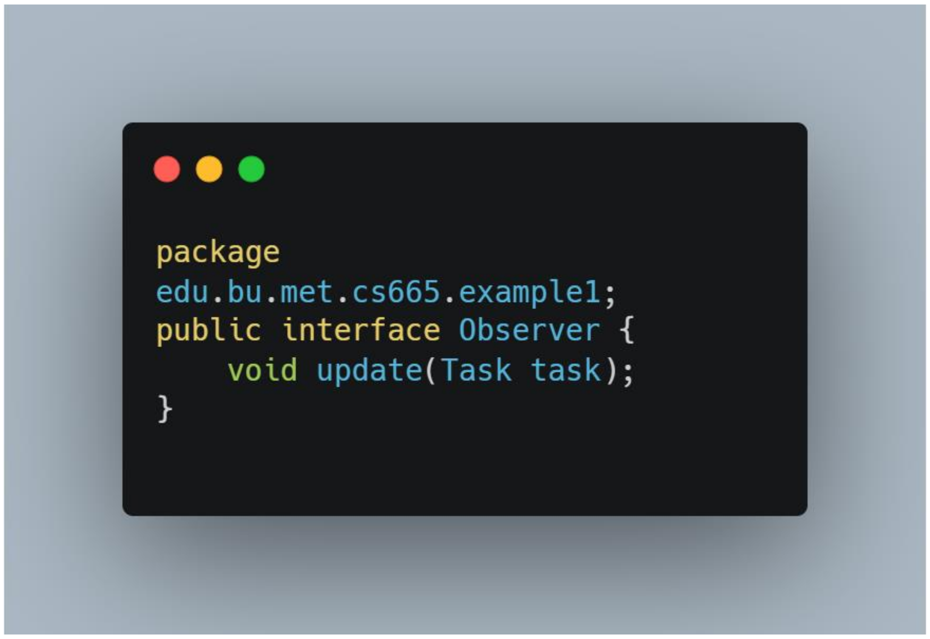
Name: Tianfu Xu Course: CS-665 Software Designs & Patterns Date: 10/07/2023

Observable.java



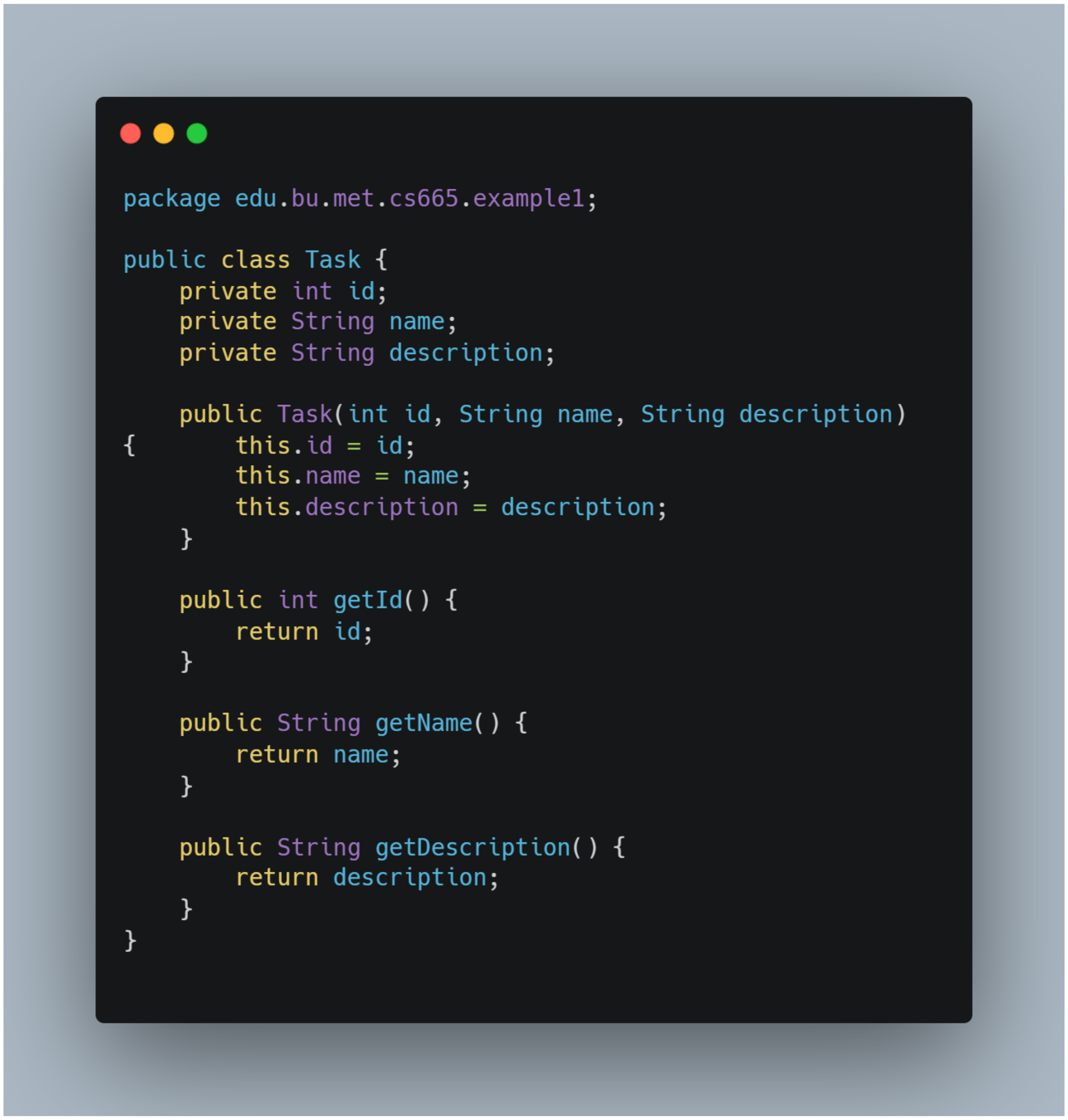
```java
package edu.bu.met.cs665.example1;

public interface Observable {
    void addObserver(Observer observer);
    void removeObserver(Observer observer);
    void notifyObservers(Task task);
}
```

Observer.java



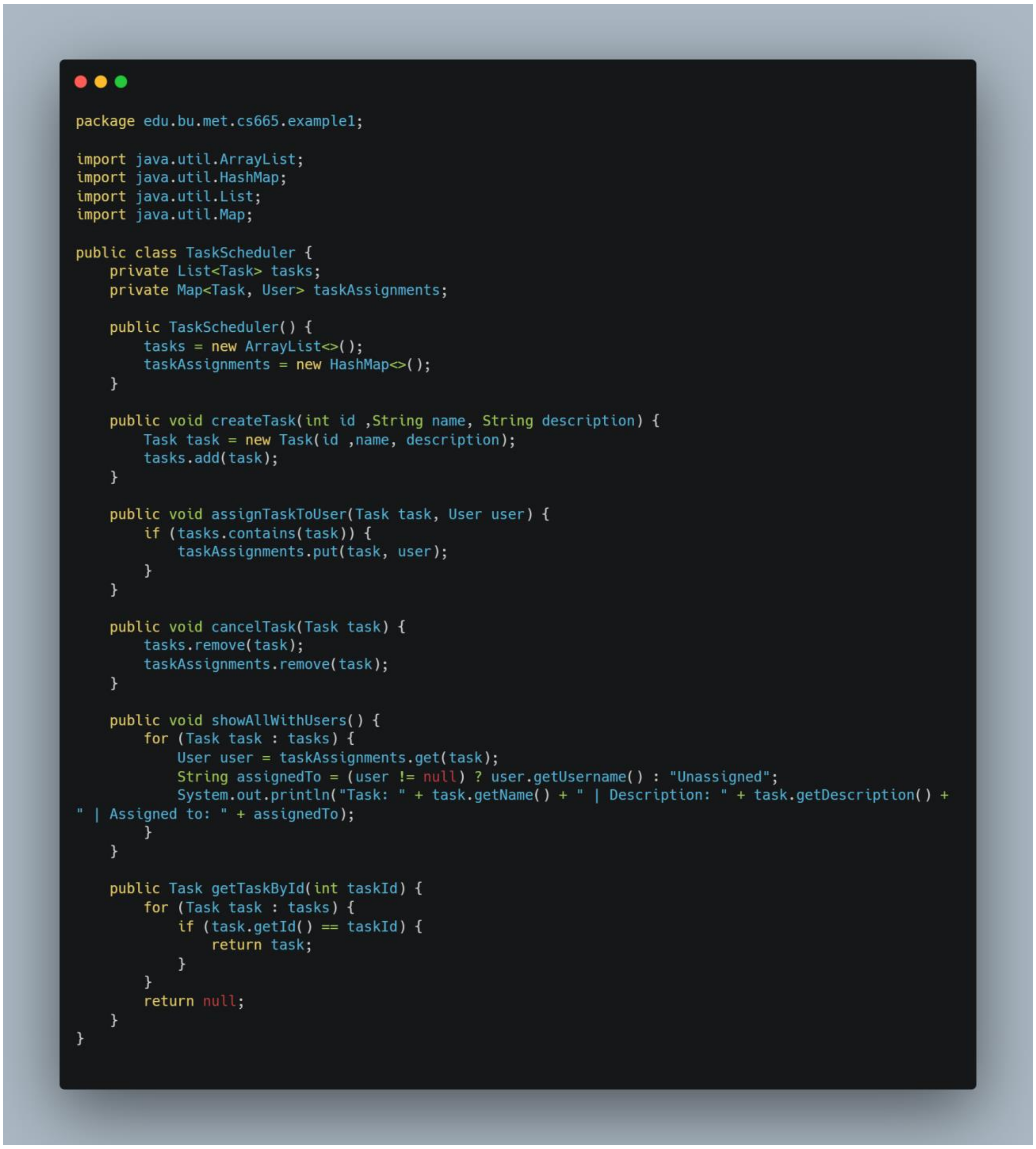
```java
package
edu.bu.met.cs665.example1;
public interface Observer {
    void update(Task task);
}
```

Task.java



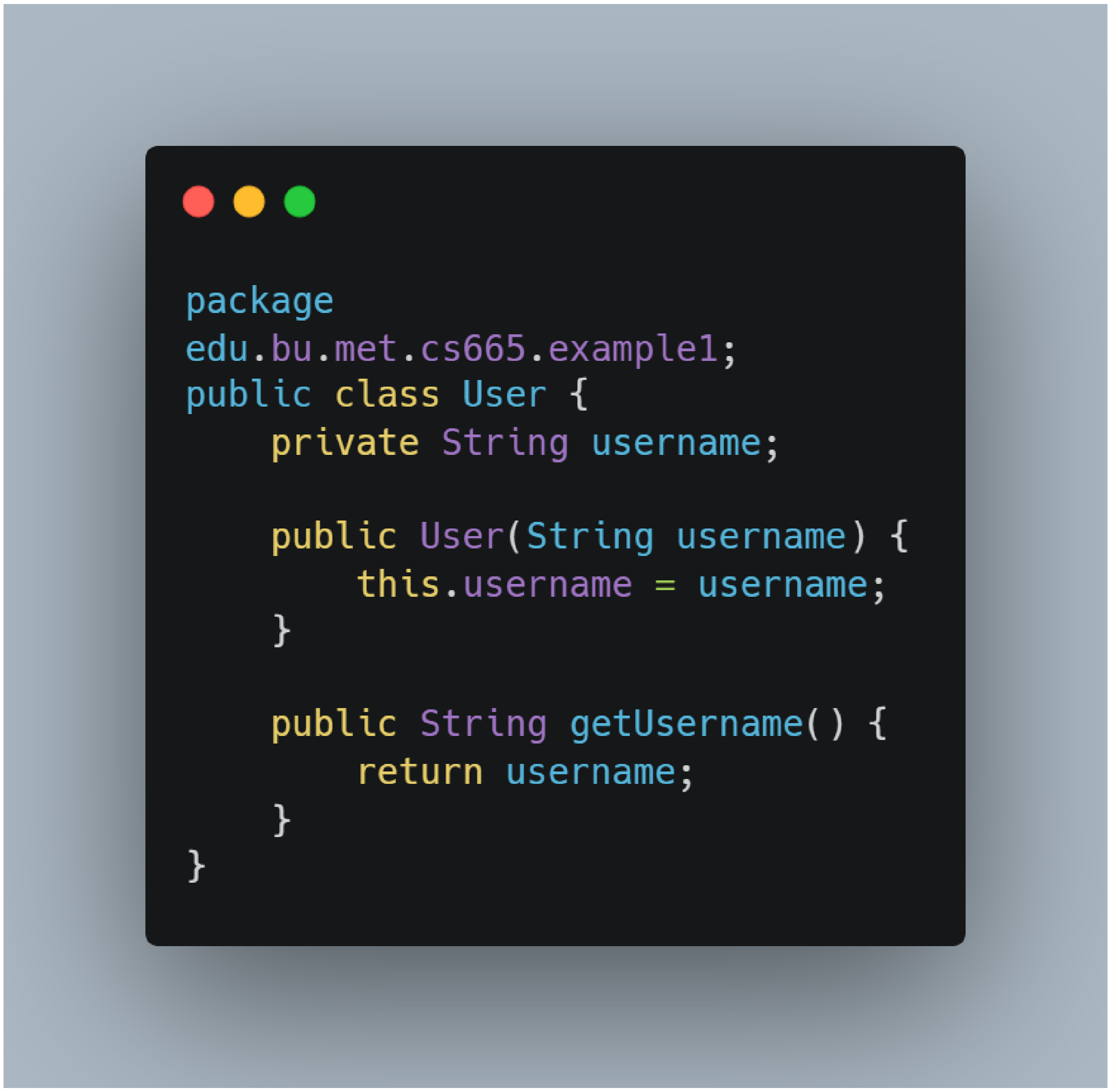
```java
package edu.bu.met.cs665.example1;

public class Task {
    private int id;
    private String name;
    private String description;

    public Task(int id, String name, String description)
{       this.id = id;
        this.name = name;
        this.description = description;
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public String getDescription() {
        return description;
    }
}
```

TaskScheduler.java



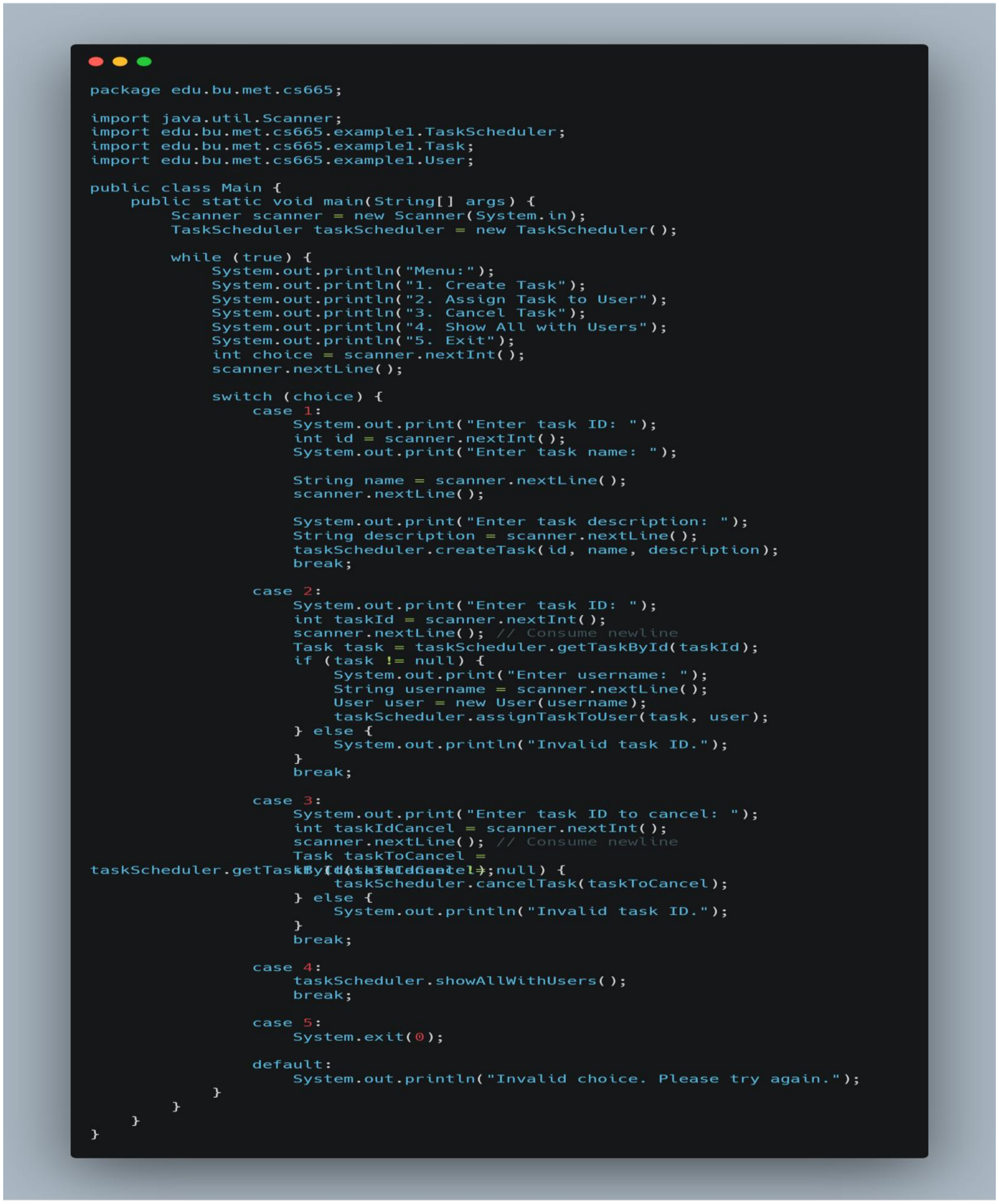
```java
package edu.bu.met.cs665.example1;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class TaskScheduler {
    private List<Task> tasks;
    private Map<Task, User> taskAssignments;

    public TaskScheduler() {
        tasks = new ArrayList<>();
        taskAssignments = new HashMap<>();
    }

    public void createTask(int id ,String name, String description) {
        Task task = new Task(id ,name, description);
        tasks.add(task);
    }

    public void assignTaskToUser(Task task, User user) {
        if (tasks.contains(task)) {
            taskAssignments.put(task, user);
        }
    }

    public void cancelTask(Task task) {
        tasks.remove(task);
        taskAssignments.remove(task);
    }

    public void showAllWithUsers() {
        for (Task task : tasks) {
            User user = taskAssignments.get(task);
            String assignedTo = (user != null) ? user.getUsername() : "Unassigned";
            System.out.println("Task: " + task.getName() + " | Description: " + task.getDescription() +
" | Assigned to: " + assignedTo);
        }
    }

    public Task getTaskById(int taskId) {
        for (Task task : tasks) {
            if (task.getId() == taskId) {
                return task;
            }
        }
        return null;
    }
}
```

User.java



```java
package
edu.bu.met.cs665.example1;
public class User {
    private String username;

    public User(String username) {
        this.username = username;
    }

    public String getUsername() {
        return username;
    }
}
```

Main.java



```java
package edu.bu.met.cs665;

import java.util.Scanner;
import edu.bu.met.cs665.example1.TaskScheduler;
import edu.bu.met.cs665.example1.Task;
import edu.bu.met.cs665.example1.User;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        TaskScheduler taskScheduler = new TaskScheduler();

        while (true) {
            System.out.println("Menu:");
            System.out.println("1. Create Task");
            System.out.println("2. Assign Task to User");
            System.out.println("3. Cancel Task");
            System.out.println("4. Show All with Users");
            System.out.println("5. Exit");
            int choice = scanner.nextInt();
            scanner.nextLine();

            switch (choice) {
                case 1:
                    System.out.print("Enter task ID: ");
                    int id = scanner.nextInt();
                    System.out.print("Enter task name: ");

                    String name = scanner.nextLine();
                    scanner.nextLine();

                    System.out.print("Enter task description: ");
                    String description = scanner.nextLine();
                    taskScheduler.createTask(id, name, description);
                    break;

                case 2:
                    System.out.print("Enter task ID: ");
                    int taskId = scanner.nextInt();
                    scanner.nextLine(); // Consume newline
                    Task task = taskScheduler.getTaskById(taskId);
                    if (task != null) {
                        System.out.print("Enter username: ");
                        String username = scanner.nextLine();
                        User user = new User(username);
                        taskScheduler.assignTaskToUser(task, user);
                    } else {
                        System.out.println("Invalid task ID.");
                    }
                    break;

                case 3:
                    System.out.print("Enter task ID to cancel: ");
                    int taskIdCancel = scanner.nextInt();
                    scanner.nextLine(); // Consume newline
                    Task taskToCancel =
taskScheduler.getTaskById(taskIdCancel);
                    if (taskToCancel != null) {
                        taskScheduler.cancelTask(taskToCancel);
                    } else {
                        System.out.println("Invalid task ID.");
                    }
                    break;

                case 4:
                    taskScheduler.showAllWithUsers();
                    break;

                case 5:
                    System.exit(0);

                default:
                    System.out.println("Invalid choice. Please try again.");
            }
        }
    }
}
```

Flexibility:

Adding New Object Types: The implementation is fairly flexible in terms of adding new object types. If you want to introduce a new type of object (e.g., adding a new attribute or behavior), you can create a new class for it (e.g., NewObjectType) similar to Task and User. You would then integrate it into the existing classes (e.g., TaskScheduler) by adding appropriate methods and data structures.

Removing Object Types: If you want to remove an existing object type, you can simply remove the associated class (e.g., Task or User) and remove references to it in the TaskScheduler class.

Simplicity and Understandability:

The code is structured in a clear and organized manner. Each class has a specific responsibility, making it easy for others to understand and maintain.

Meaningful method and variable names make the code self-explanatory, reducing the need for extensive comments.

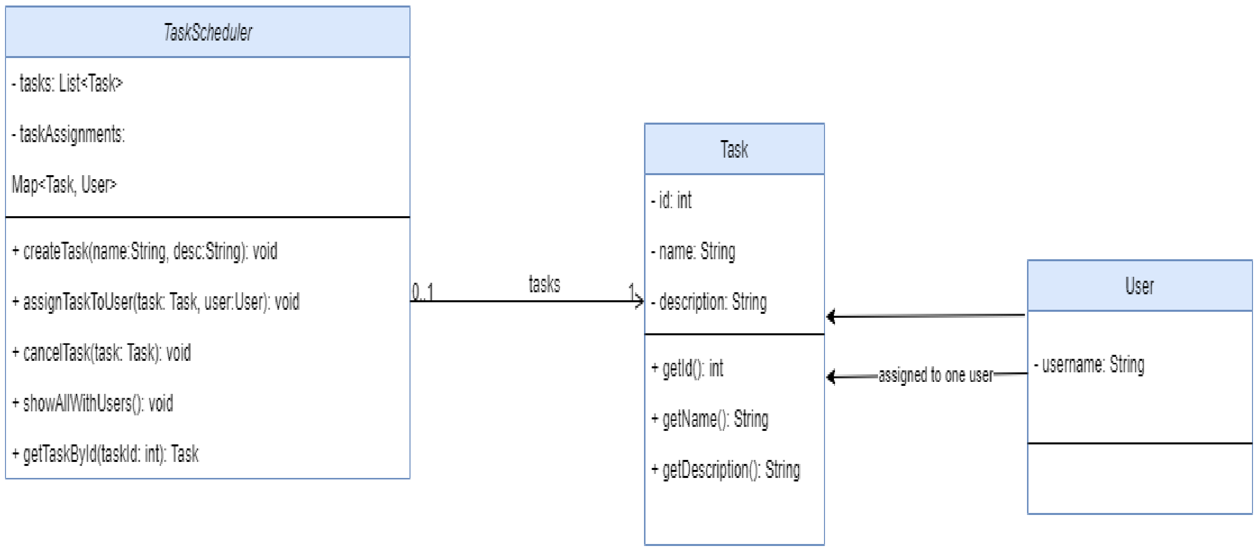
Avoidance of Duplicated Code:

The implementation aims to follow the DRY (Don't Repeat Yourself) principle. For example, the TaskScheduler class encapsulates the common functionality related to tasks and users, avoiding the need to duplicate this code elsewhere.

Additionally, methods like createTask, assignTaskToUser, cancelTask, and showAllWithUsers handle specific tasks, reducing redundancy.

Design Patterns:

The implementation utilizes a basic form of the Observer pattern. When a Task is created, it's added to the list of tasks in the TaskScheduler. This allows the TaskScheduler to observe and manage tasks. `TaskScheduler`. This allows the `TaskScheduler` to observe and manage tasks.


TaskScheduler
- tasks: List<Task>
- taskAssignments: Map<Task, User>
+ createTask(name:String, desc:String): void
+ assignTaskToUser(task: Task, user:User): void
+ cancelTask(task: Task): void
+ showAllWithUsers(): void
+ getTaskById(taskId: int): Task
0..1
tasks
1
Task
- id: int
- name: String
- description: String
+ getId(): int
+ getName(): String
+ getDescription(): String
assigned to one user
User
- username: String

Explanation:

TaskScheduler:

<u>Attributes:</u>

tasks: List<Task>

taskAssignments: Map<Task, User>

<u>Methods:</u>

createTask(name: String, description: String): void

assignTaskToUser(task: Task, user: User): void

cancelTask(task: Task): void

showAllWithUsers(): void

getTaskById(taskId: int): Task

Task:

<u>Attributes:</u>

id: int

name: String

description: String

<u>Methods:</u>

getId(): int

getName(): String

getDescription(): String

User:

<u>Attributes:</u>

username: String

<u>Methods:</u>

getUsername(): String

<u>Relationships:</u>

TaskScheduler has a list of tasks (1 to many)

Task has an assigned user (1 to 1)

Task can be assigned to one user