DOCUMENTATION WORKSHOP-2 PROJECT OBJECT-ORIENTED PROGRAMMING: NOTION (NOTES APP)

Juan Carlos Córdoba Asprilla – 20242020047

Sebastián Camilo Sánchez Cárdenas- 20242020086

Marzo 01 de 2025

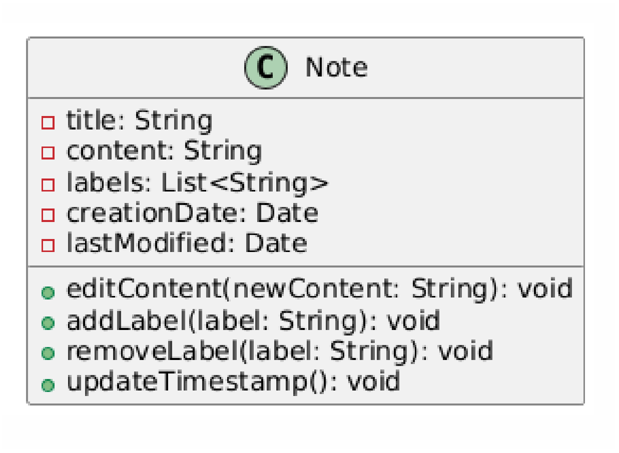
UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS

Object-oriented Programming

Ing. Carlos Andrés Sierra

CONCEPTUAL DESING UPDATES:

1. Addition of the Note Class



Reason:

Although the Notes class referenced a list of Note objects (notesList: List<Note>), the Note class itself was not explicitly defined. It has been created to represent a single note, fulfilling key user stories and functional requirements related to note creation, editing, tagging, and tracking modification times.

Attributes:

- title: String
- content: String
- labels: List<String>
- creationDate: Date
- lastModified: Date

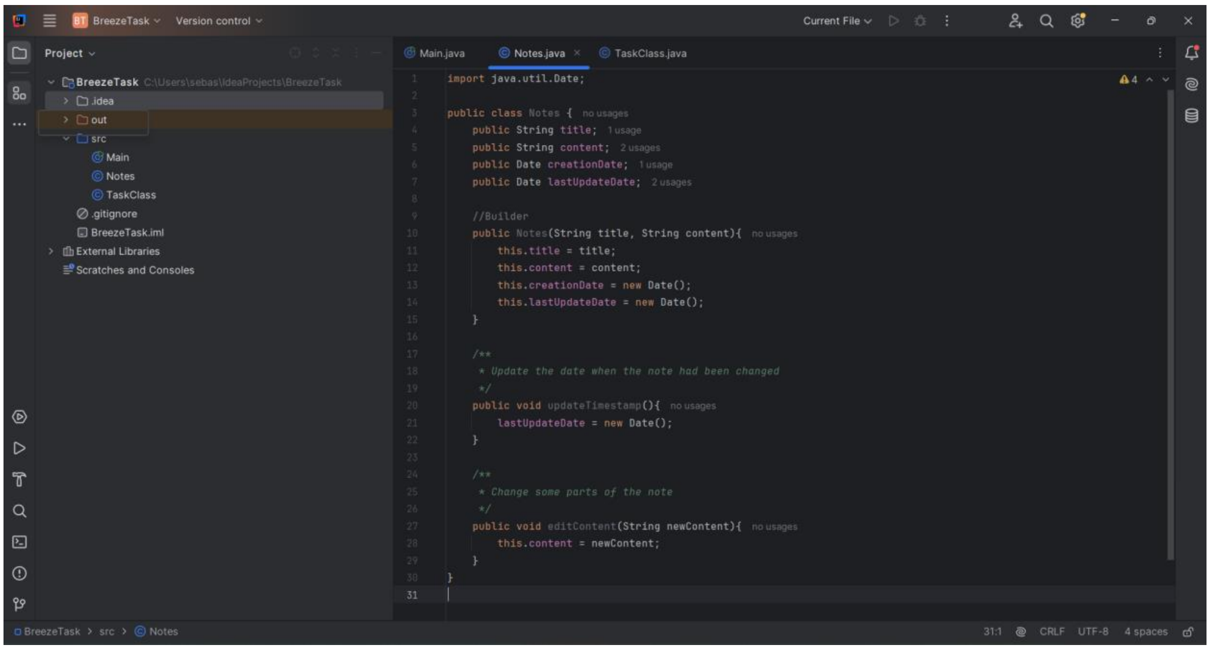
Methods:

- editContent(newContent: String): void
- addLabel(label: String): void
- removeLabel(label: String): void
- updateTimestamp(): void

Relationships:

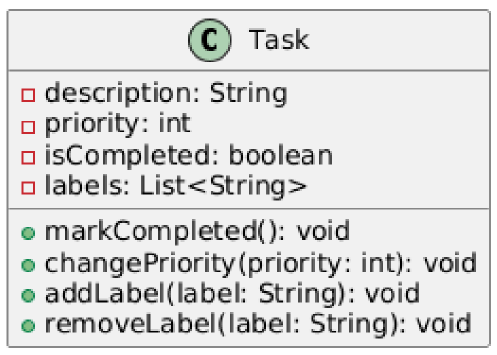
Notes --> Note

Note --> Storage



```java
import java.util.Date;

public class Notes {
    public String title;
    public String content;
    public Date creationDate;
    public Date lastUpdateDate;

    //Builder
    public Notes(String title, String content){
        this.title = title;
        this.content = content;
        this.creationDate = new Date();
        this.lastUpdateDate = new Date();
    }

    /**
     * Update the date when the note had been changed
     */
    public void updateTimestamp(){
        lastUpdateDate = new Date();
    }

    /**
     * Change some parts of the note
     */
    public void editContent(String newContent){
        this.content = newContent;
    }
}
```

2. Addition of the Task Class



Reason:
The Tasks class was already listed as managing multiple tasks, but no definition existed for the individual Task entity. This class was added to model a single task with its priority, completion status, and labels—supporting user stories that involve goal tracking, task organization, and filtering.

Attributes:

- description: String
- priority: int (ejem., 1 = high, 2 = medium, 3 = low)
- isCompleted: boolean
- labels: List<String>

Methods:

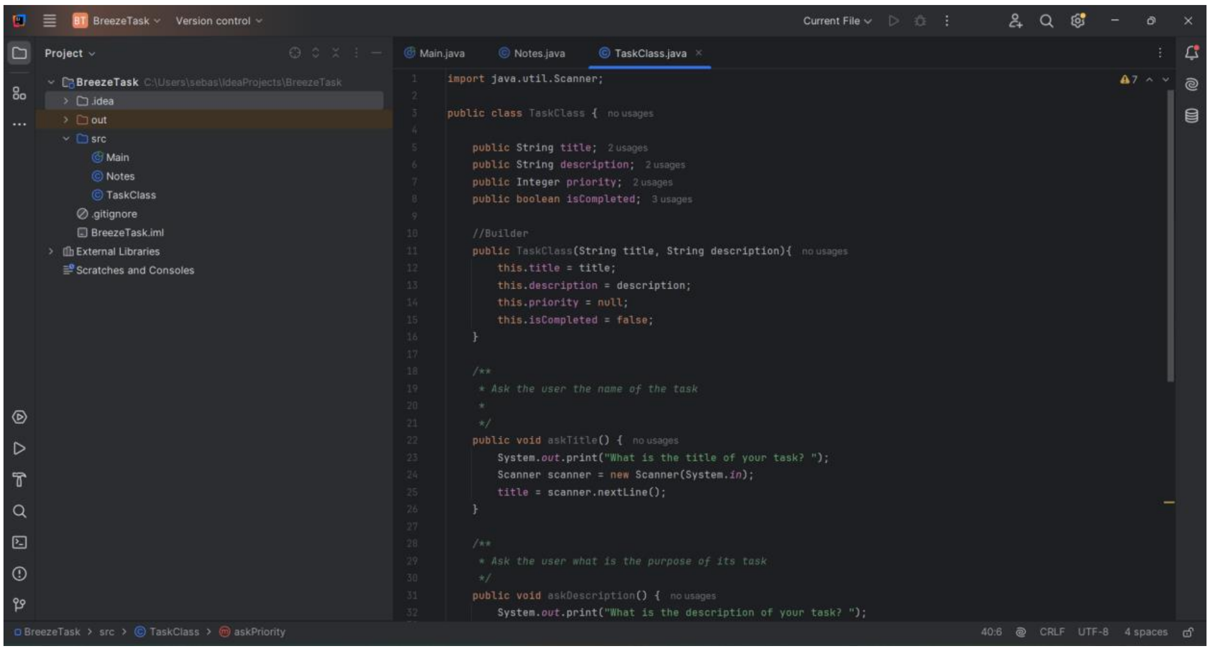
- markCompleted(): void

- changePriority(priority: int): void
- addLabel(label: String): void
- removeLabel(label: String): void

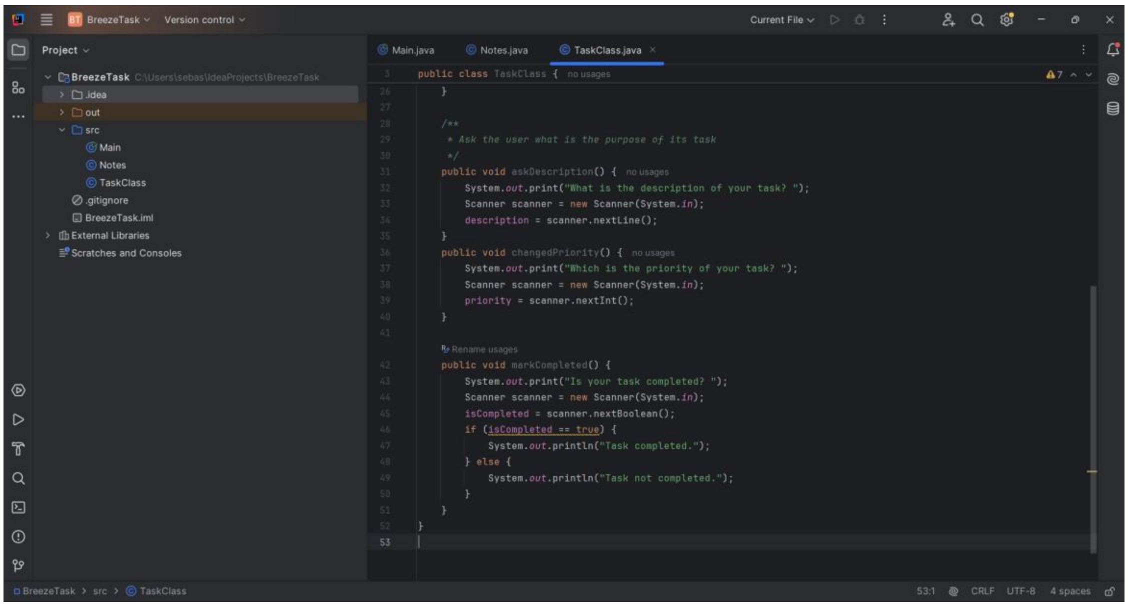
Relationships:

Tasks --> Task

Task --> Storage



```java
import java.util.Scanner;

public class TaskClass {

    public String title;
    public String description;
    public Integer priority;
    public boolean isCompleted;

    //Builder
    public TaskClass(String title, String description){
        this.title = title;
        this.description = description;
        this.priority = null;
        this.isCompleted = false;
    }

    /**
     * Ask the user the name of the task
     *
     */
    public void askTitle() {
        System.out.print("What is the title of your task? ");
        Scanner scanner = new Scanner(System.in);
        title = scanner.nextLine();
    }

    /**
     * Ask the user what is the purpose of its task
     */
    public void askDescription() {
        System.out.print("What is the description of your task? ");
```



```java
public class TaskClass {
    }

    /**
     * Ask the user what is the purpose of its task
     */
    public void askDescription() {
        System.out.print("What is the description of your task? ");
        Scanner scanner = new Scanner(System.in);
        description = scanner.nextLine();
    }
    public void changedPriority() {
        System.out.print("Which is the priority of your task? ");
        Scanner scanner = new Scanner(System.in);
        priority = scanner.nextInt();
    }

    public void markCompleted() {
        System.out.print("Is your task completed? ");
        Scanner scanner = new Scanner(System.in);
        isCompleted = scanner.nextBoolean();
        if (isCompleted == true) {
            System.out.println("Task completed.");
        } else {
            System.out.println("Task not completed.");
        }
    }
}
```

DIAGRAMS

ACTIVITY DIAGRAM - UML CLASS DIAGRAM - SECUENCE DIAGRAM

Note: The diagrams shown below use classes with conceptual (yet to be defined) attributes and methods to establish their respective connections. In the future, the structure of these methods and the necessary attributes to include or exclude will be reviewed to ensure the application functions are as intended.

APPLICATION OF OOP CONCEPTS

At this stage of the application's development, the use of object-oriented programming principles is not yet fully defined or implemented, as many functionalities are focused on direct user interaction with specific modules.

Regarding **encapsulation**, protecting the user's information has been considered essential, especially since this is a local application — without online connectivity — meaning all data will be stored directly on the user's device. Therefore, appropriate access modifiers are planned to restrict direct access to attributes and ensure that any modifications are handled through well-defined public methods.

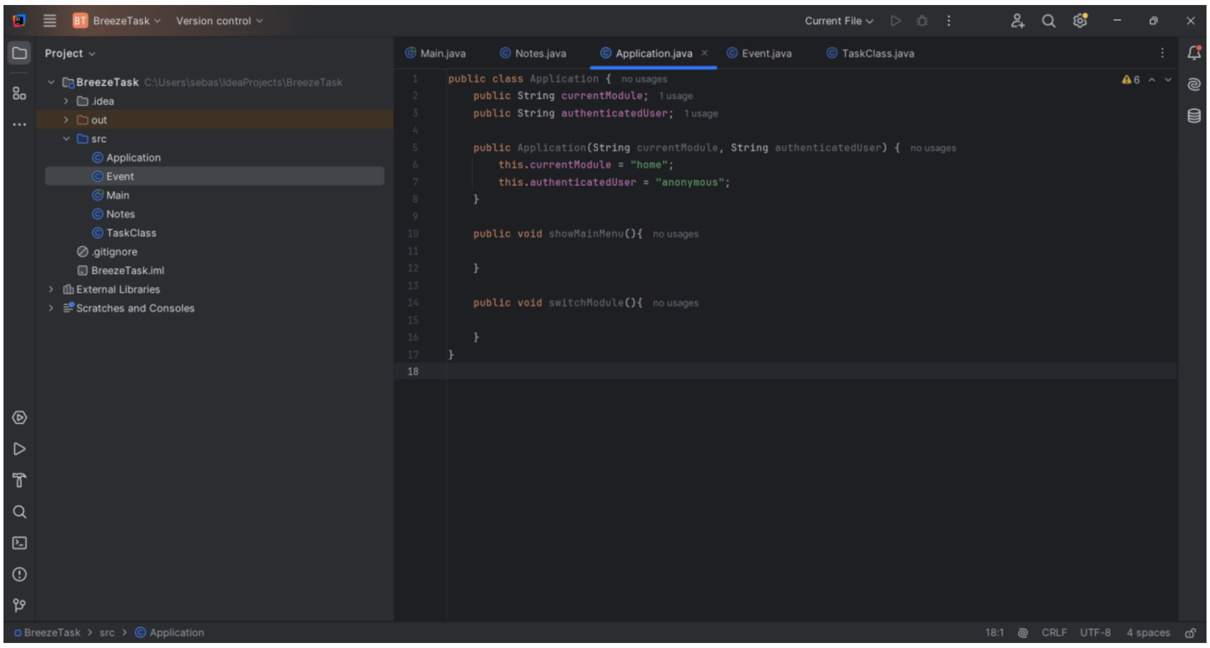
Concerning **inheritance**, the possibility of creating a base class called Module has been considered, from which components such as calendar, notes, tasks, and reminders could be derived. However, this decision is still under evaluation, as it must be determined whether it truly adds value or introduces unnecessary complexity into the design.

Lastly, regarding **polymorphism**, its implementation is not currently anticipated, as each class's responsibilities are clearly defined and do not require flexible or alternative behaviors. Each module operates independently and fulfills specific tasks, making the use of this concept currently unnecessary.

PACKAGE ESTRUCTURE

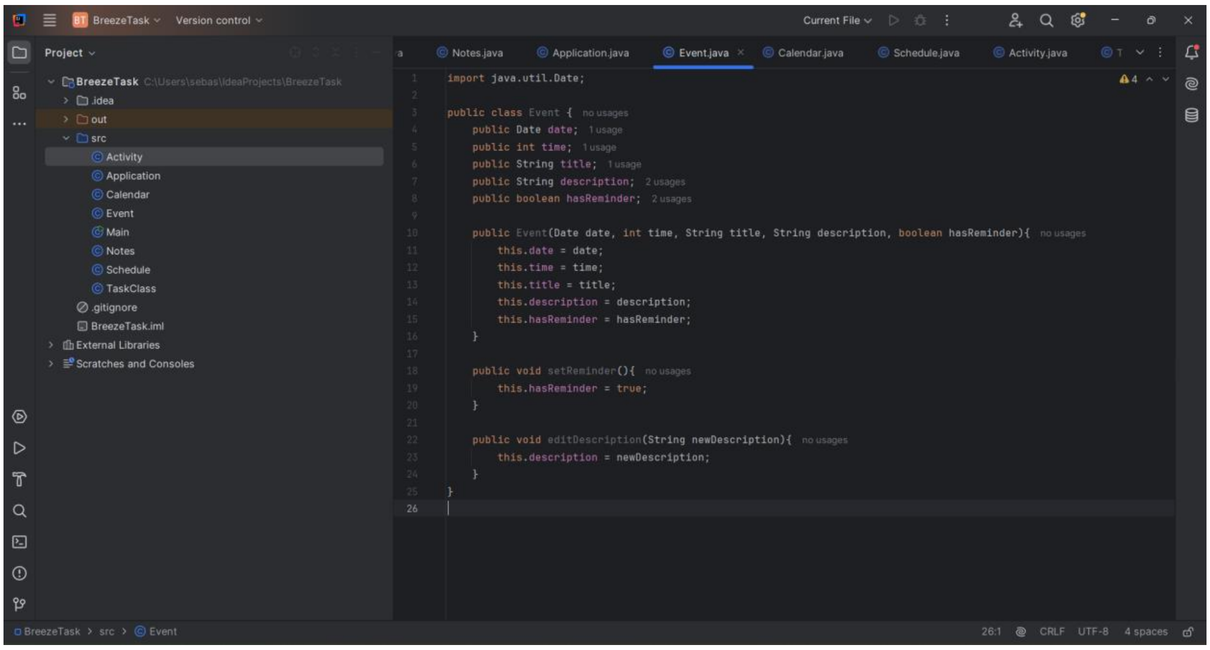
WORK IN PROGRESS

Application



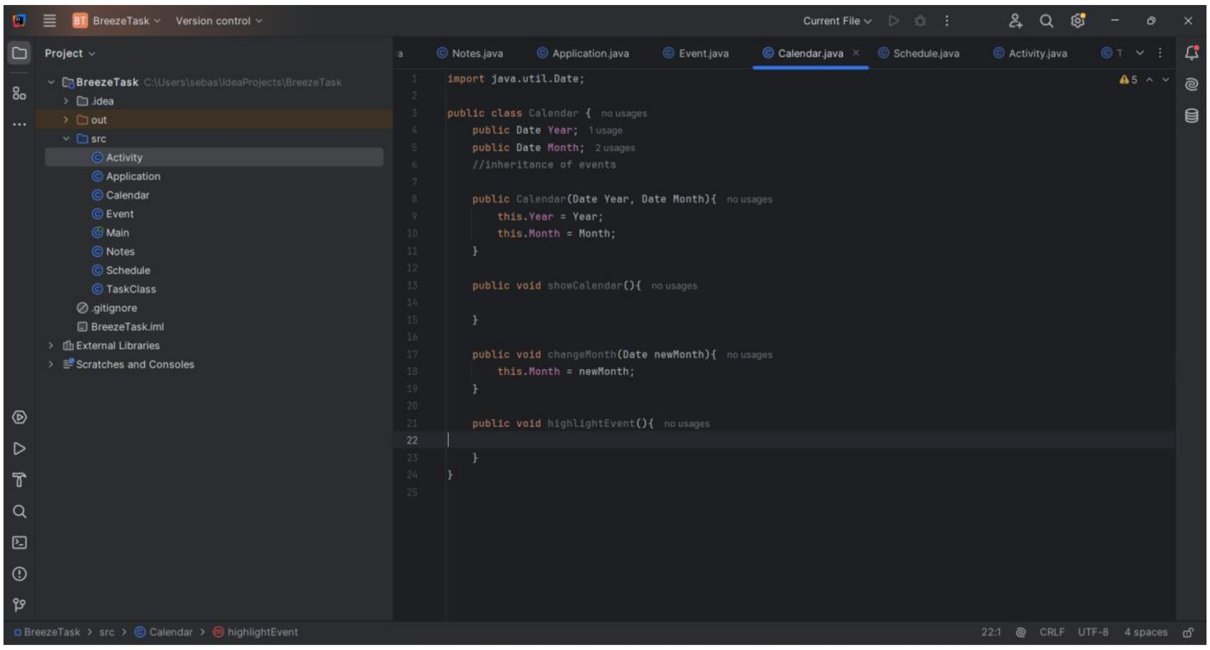
```java
public class Application {
    public String currentModule;
    public String authenticatedUser;

    public Application(String currentModule, String authenticatedUser) {
        this.currentModule = "home";
        this.authenticatedUser = "anonymous";
    }

    public void showMainMenu(){

    }

    public void switchModule(){

    }
}
```

Event



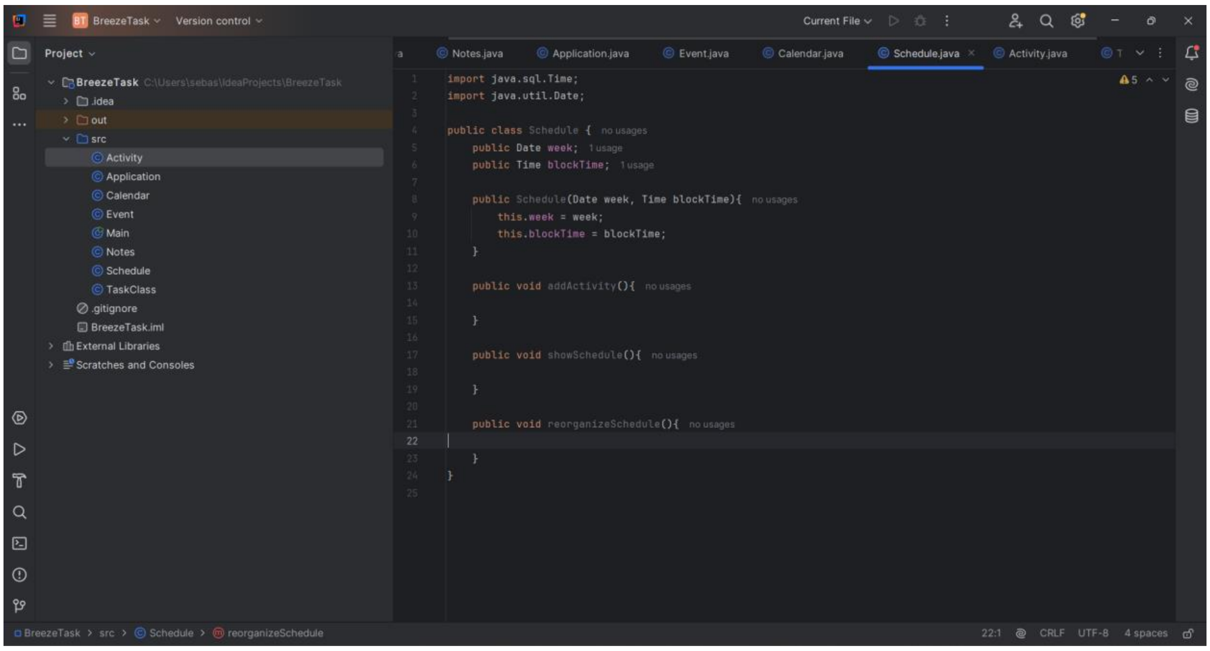
```java
import java.util.Date;

public class Event {
    public Date date;
    public int time;
    public String title;
    public String description;
    public boolean hasReminder;

    public Event(Date date, int time, String title, String description, boolean hasReminder){
        this.date = date;
        this.time = time;
        this.title = title;
        this.description = description;
        this.hasReminder = hasReminder;
    }

    public void setReminder(){
        this.hasReminder = true;
    }

    public void editDescription(String newDescription){
        this.description = newDescription;
    }
}
```

Calendar



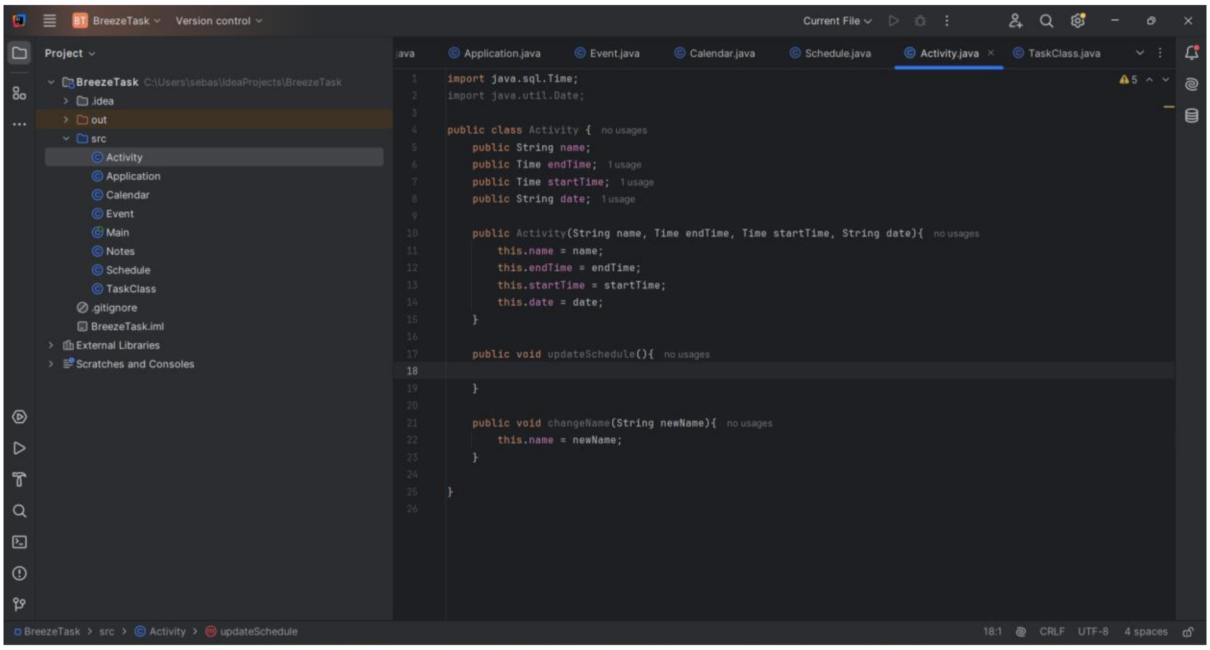
```java
import java.util.Date;

public class Calendar {
    public Date Year;
    public Date Month;
    //inheritance of events

    public Calendar(Date Year, Date Month){
        this.Year = Year;
        this.Month = Month;
    }

    public void showCalendar(){

    }

    public void changeMonth(Date newMonth){
        this.Month = newMonth;
    }

    public void highlightEvent(){

    }
}
```

Schedule



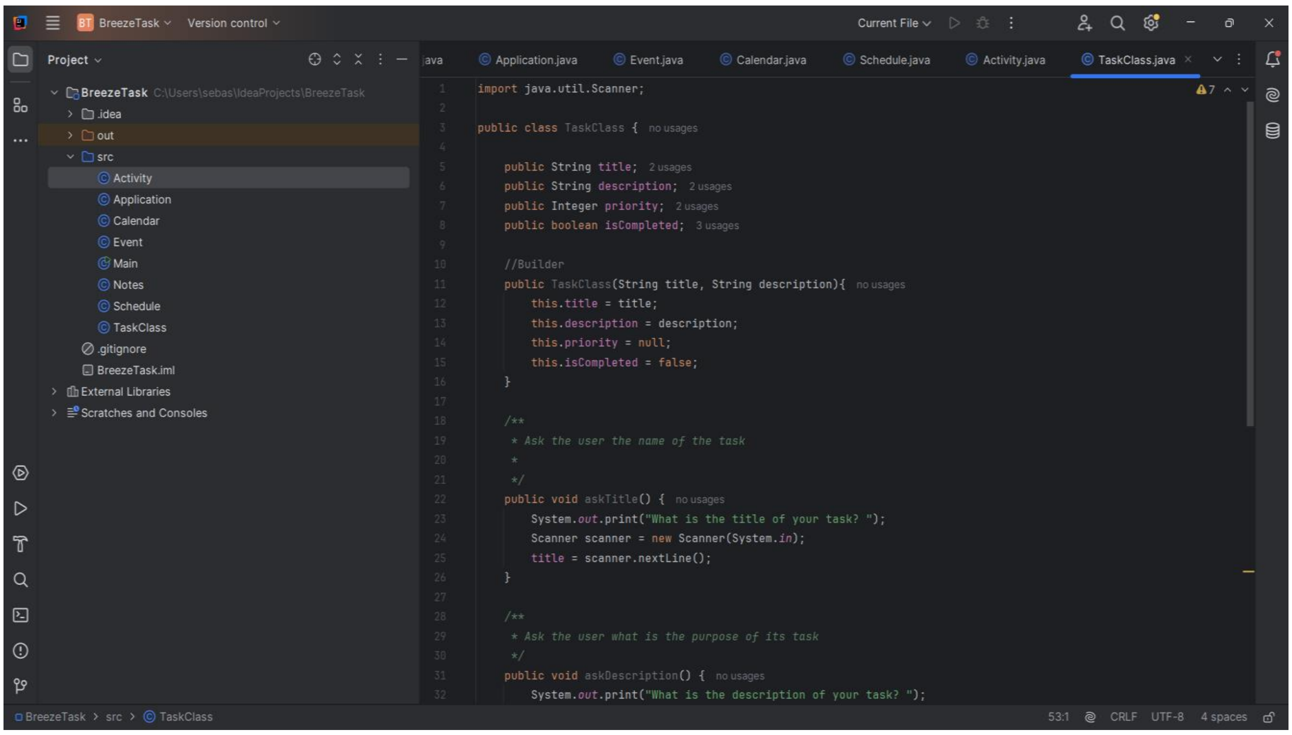
```java
import java.sql.Time;
import java.util.Date;

public class Schedule {
    public Date week;
    public Time blockTime;

    public Schedule(Date week, Time blockTime){
        this.week = week;
        this.blockTime = blockTime;
    }

    public void addActivity(){

    }

    public void showSchedule(){

    }

    public void reorganizeSchedule(){

    }
}
```

Activity

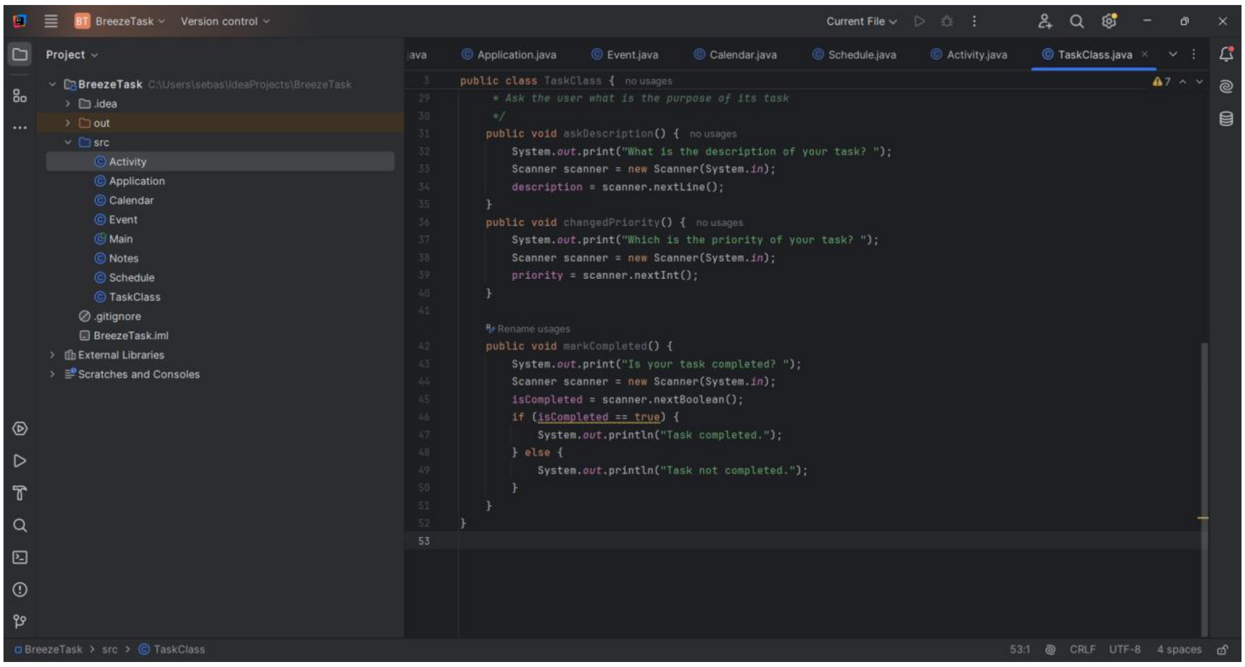


```java
import java.sql.Time;
import java.util.Date;

public class Activity {
    public String name;
    public Time endTime;
    public Time startTime;
    public String date;

    public Activity(String name, Time endTime, Time startTime, String date){
        this.name = name;
        this.endTime = endTime;
        this.startTime = startTime;
        this.date = date;
    }

    public void updateSchedule(){

    }

    public void changeName(String newName){
        this.name = newName;
    }

}
```

Task



```java
import java.util.Scanner;

public class TaskClass {

    public String title;
    public String description;
    public Integer priority;
    public boolean isCompleted;

    //Builder
    public TaskClass(String title, String description){
        this.title = title;
        this.description = description;
        this.priority = null;
        this.isCompleted = false;
    }

    /**
     * Ask the user the name of the task
     *
     */
    public void askTitle() {
        System.out.print("What is the title of your task? ");
        Scanner scanner = new Scanner(System.in);
        title = scanner.nextLine();
    }

    /**
     * Ask the user what is the purpose of its task
     */
    public void askDescription() {
        System.out.print("What is the description of your task? ");
```



```java
public class TaskClass {  no usages
     * Ask the user what is the purpose of its task
     */
    public void askDescription() {  no usages
        System.out.print("What is the description of your task? ");
        Scanner scanner = new Scanner(System.in);
        description = scanner.nextLine();
    }
    public void changedPriority() {  no usages
        System.out.print("Which is the priority of your task? ");
        Scanner scanner = new Scanner(System.in);
        priority = scanner.nextInt();
    }

    Rename usages
    public void markCompleted() {
        System.out.print("Is your task completed? ");
        Scanner scanner = new Scanner(System.in);
        isCompleted = scanner.nextBoolean();
        if (isCompleted == true) {
            System.out.println("Task completed.");
        } else {
            System.out.println("Task not completed.");
        }
    }
}
```