**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра ІПІ**

**Звіт**

з лабораторної роботи № 5 з дисципліни

«Основи програмування 2.

Модульне програмування»

Варіант 25

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-15 Плугатирьов Дмитро Валерійович ійович*

**Перевірив**

(прізвище, ім'я, по батькові)

*Вєчерковська Анастасія Сергіївна*

Київ 2022

**УСПАДКУВАННЯ ТА ПОЛІМОРФІЗМ**

*Мета роботи* – вивчити механізми створення і використання класів та об’єктів.

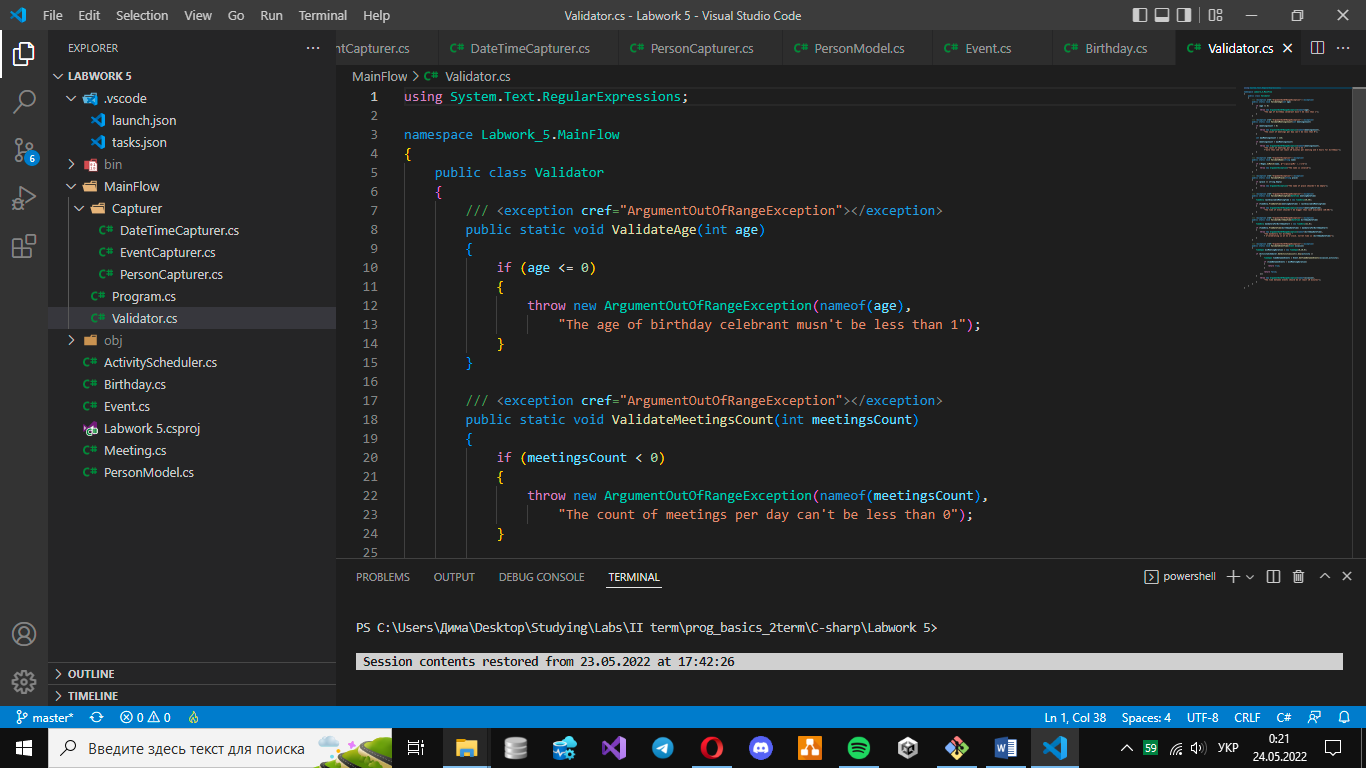
**Варіант 25**

**Завдання**

Створити клас «Подія», який містить дату і час певної події, а також методи обчислення часу, що залишився до початку події. На основі цього класу створити класи-нащадки «День народження», який містить ПІБ іменинника, його вік, місце проведення свята, та «Зустріч», який містить ПІБ людини, з якою призначена зустріч, і місце зустрічі. Створити розклад активностей особи на конкретну дату, який включає n зустрічей і одне святкування дня народження. Визначити останню заплановану зустріч в цей день і інтервал часу від її закінчення до початку святкування дня народження.

**Код програми**

**С#**



namespace Labwork\_5.MainFlow.Capturer

{

    public class DateTimeCapturer

    {

        public static TimeOnly CaptureTime()

        {

            TimeOnly time = new TimeOnly();

            bool valueIsValid = false;

            while (!valueIsValid)

            {

                valueIsValid = true;

                try

                {

                    System.Console.Write("Please, enter the time: ");

                    time = TimeOnly.Parse(Console.ReadLine());

                }

                catch (FormatException ex)

                {

                    System.Console.WriteLine(ex.Message);

                    valueIsValid = false;

                }

            }

            return time;

        }

        public static DateOnly CaptureDate()

        {

            DateOnly date = new DateOnly();

            bool valueIsValid = false;

            while (!valueIsValid)

            {

                valueIsValid = true;

                try

                {

                    System.Console.Write("Please, enter the date: ");

                    date = DateOnly.Parse(Console.ReadLine());

                }

                catch (FormatException ex)

                {

                    System.Console.WriteLine(ex.Message);

                    valueIsValid = false;

                }

            }

            return date;

        }

    }

}

namespace Labwork\_5.MainFlow.Capturer

{

    public class EventCapturer

    {

        private int \_meetingsCount;

        /// <exception cref="ArgumentOutOfRangeException"></exception>

        public int MeetingsCount

        {

            get => \_meetingsCount;

            set

            {

                Validator.ValidateMeetingsCount(value);

                \_meetingsCount = value;

            }

        }

        public EventCapturer(int meetingsCount)

        {

            MeetingsCount = meetingsCount;

        }

        public EventCapturer()

        {

        }

        public List<Event> CaptureEvents()

        {

            do

            {

                try

                {

                    MeetingsCount--;

                    ActivityScheduler.AddActivity(CaptureMeeting());

                    if (MeetingsCount == 0)

                    {

                        ActivityScheduler.AddActivity(CaptureBirthday());

                    }

                }

                catch (ArgumentOutOfRangeException ex)

                {

                    System.Console.WriteLine(ex.Message);

                    MeetingsCount++;

                }

            } while (MeetingsCount > 0);

            return ActivityScheduler.GetActivitiesList();

        }

        public int CaptureMaxMeetingsCount()

        {

            bool exceptionIsThrown = true;

            while (exceptionIsThrown)

            {

                System.Console.Write("Please, enter maximal count of meetings per day: ");

                exceptionIsThrown = false;

                try

                {

                    MeetingsCount = int.Parse(Console.ReadLine());

                }

                catch (FormatException)

                {

                    System.Console.WriteLine("The entered value isn't a number");

                    exceptionIsThrown = true;

                }

                catch (ArgumentOutOfRangeException ex)

                {

                    System.Console.WriteLine(ex.Message);

                    exceptionIsThrown = true;

                }

            }

            return MeetingsCount;

        }

        private static Meeting CaptureMeeting()

        {

            Meeting meeting = new Meeting();

            bool exceptionIsThrown = true;

            while (exceptionIsThrown)

            {

                System.Console.WriteLine("You are forming a meeting:");

                exceptionIsThrown = false;

                try

                {

                    System.Console.WriteLine("Please, identificate the person to meet:");

                    PersonCapturer.CapturePerson(meeting);

                    CapturePlace(meeting);

                    meeting.DateTime = meeting.DateTime.Add(DateTimeCapturer.CaptureTime().ToTimeSpan());

                    Program.PrintDashLine();

                }

                catch (ArgumentException ex)

                {

                    System.Console.WriteLine(ex.Message);

                    exceptionIsThrown = true;

                }

            }

            return meeting;

        }

        private static Birthday CaptureBirthday()

        {

            Birthday birthday = new Birthday();

            bool exceptionIsThrown = true;

            while (exceptionIsThrown)

            {

                System.Console.WriteLine("You are forming a birthday:");

                exceptionIsThrown = false;

                try

                {

                    System.Console.WriteLine("Please, identificate the celebrant:");

                    PersonCapturer.CapturePerson(birthday);

                    CapturePlace(birthday);

                    birthday.DateTime = birthday.DateTime.Add(DateTimeCapturer.CaptureTime().ToTimeSpan());

                    Program.PrintDashLine();

                }

                catch (ArgumentOutOfRangeException ex)

                {

                    System.Console.WriteLine(ex.Message);

                    exceptionIsThrown = true;

                }

                catch (ArgumentException ex)

                {

                    System.Console.WriteLine(ex.Message);

                    exceptionIsThrown = true;

                }

            }

            return birthday;

        }

        private static string CapturePlace(Event activity)

        {

            string place = default;

            bool exceptionIsThrown = true;

            while (exceptionIsThrown)

            {

                System.Console.Write("Please, enter the place: ");

                exceptionIsThrown = false;

                try

                {

                    if (activity is Birthday birthday)

                    {

                        birthday.CelebrationPlace = place = Console.ReadLine();

                    }

                    else if (activity is Meeting meeting)

                    {

                        meeting.Place = place = Console.ReadLine();

                    }

                }

                catch (ArgumentException ex)

                {

                    System.Console.WriteLine(ex.Message);

                    exceptionIsThrown = true;

                }

            }

            return place;

        }

    }

}

namespace Labwork\_5.MainFlow.Capturer

{

    public class PersonCapturer

    {

        private static int CaptureAge(Birthday birthday)

        {

            int age = default;

            bool exceptionIsThrown = true;

            while (exceptionIsThrown)

            {

                System.Console.Write("Please, enter the celebrant's age: ");

                exceptionIsThrown = false;

                try

                {

                    birthday.CelebrantsAge = age = int.Parse(Console.ReadLine());

                }

                catch (FormatException)

                {

                    System.Console.WriteLine("The entered value isn't a number");

                    exceptionIsThrown = true;

                }

                catch (ArgumentOutOfRangeException ex)

                {

                    System.Console.WriteLine(ex.Message);

                    exceptionIsThrown = true;

                }

            }

            return age;

        }

        public static PersonModel CapturePerson(Event activity)

        {

            PersonModel person = new PersonModel();

            bool exceptionIsThrown = true;

            while (exceptionIsThrown)

            {

                exceptionIsThrown = false;

                try

                {

                    System.Console.Write("Please, enter the first name: ");

                    person.FirstName = Console.ReadLine();

                    System.Console.Write("Please, enter the second name: ");

                    person.SecondName = Console.ReadLine();

                    System.Console.Write("Please, enter the patronymic: ");

                    person.Patronymic = Console.ReadLine();

                    if (activity is Birthday birthday)

                    {

                        CaptureAge(birthday);

                        birthday.Celebrant = person;

                    }

                    else if (activity is Meeting meeting)

                    {

                        meeting.PersonToMeet = person;

                    }

                }

                catch (ArgumentException ex)

                {

                    System.Console.WriteLine(ex.Message);

                    exceptionIsThrown = true;

                }

            }

            return person;

        }

    }

}

using Labwork\_5.MainFlow.Capturer;

namespace Labwork\_5.MainFlow

{

    class Program

    {

        static void Main(string[] args)

        {

            System.Console.WriteLine("Please, enter the concrete date when you are free from chores: ");

            DateOnly concreteDate = DateTimeCapturer.CaptureDate();

            EventCapturer eventCapturer = new EventCapturer();

            eventCapturer.MeetingsCount = eventCapturer.CaptureMaxMeetingsCount();

            List<Event> activities = eventCapturer.CaptureEvents();

            ActivityScheduler.AssignDateToEvents(concreteDate);

            PrintEvents();

            System.Console.WriteLine("The last meeting of the day:");

            Meeting lastMeeting = ActivityScheduler.GetLatestMeeting();

            System.Console.WriteLine(lastMeeting);

            Birthday birthday = GetBirthdayFromList(activities);

            System.Console.Write("Period of time between last meeting and birthday: ");

            Console.WriteLine(Event.GetTimeBetweenEvents(birthday,lastMeeting));

        }

        static Birthday GetBirthdayFromList(List<Event> activities)

        {

            return activities.FirstOrDefault(activity => activity is Birthday) as Birthday;

        }

        public static void PrintDashLine()

        {

            System.Console.WriteLine(new string('-', 60));

        }

        static void PrintEvents()

        {

            PrintDashLine();

            System.Console.WriteLine("The captured activities are:");

            foreach (Event activity in ActivityScheduler.GetActivitiesList())

            {

                if (activity is Meeting meeting)

                {

                    System.Console.WriteLine($"{meeting.GetType().Name} - {meeting}");

                }

                else if (activity is Birthday birthday)

                {

                    System.Console.WriteLine($"{birthday.GetType().Name} - {birthday}");

                }

            }

            PrintDashLine();

        }

    }

}

using System.Text.RegularExpressions;

namespace Labwork\_5.MainFlow

{

    public class Validator

    {

        /// <exception cref="ArgumentOutOfRangeException"></exception>

        public static void ValidateAge(int age)

        {

            if (age <= 0)

            {

                throw new ArgumentOutOfRangeException(nameof(age),

                    "The age of birthday celebrant musn't be less than 1");

            }

        }

        /// <exception cref="ArgumentOutOfRangeException"></exception>

        public static void ValidateMeetingsCount(int meetingsCount)

        {

            if (meetingsCount < 0)

            {

                throw new ArgumentOutOfRangeException(nameof(meetingsCount),

                    "The count of meetings per day can't be less than 0");

            }

            int maxMeetingsCount = 126;

            if (meetingsCount > maxMeetingsCount)

            {

                throw new ArgumentOutOfRangeException(nameof(meetingsCount),

                    "The count of meetings per day can't be "

                    +"more than 126 (at least 10 minutes per meeting and 3 hours for birthday)");

            }

        }

        /// <exception cref="ArgumentException"></exception>

        public static void ValidateName(string name)

        {

            if (!Regex.IsMatch(name, @"^[\p{L}\p{M}' \.\-]+$"))

            {

                throw new ArgumentException("The name is invalid");

            }

        }

        /// <exception cref="ArgumentException"></exception>

        public static void ValidatePlace(string place)

        {

            if (place == string.Empty)

            {

                throw new ArgumentException("The name of place shouldn't be empty");

            }

        }

        /// <exception cref="ArgumentOutOfRangeException"></exception>

        public static void ValidateMeetingTime(DateTime meetingDateTime)

        {

            TimeOnly lastAvailableMeetingTime = new TimeOnly(20,50);

            if (TimeOnly.FromDateTime(meetingDateTime) > lastAvailableMeetingTime)

            {

                throw new ArgumentOutOfRangeException(nameof(meetingDateTime),

                    "The time of event shouldn't be bigger than last available (20:50)");

            }

        }

        /// <exception cref="ArgumentOutOfRangeException"></exception>

        public static void ValidateBirthdayTime(DateTime birthdayDateTime)

        {

            TimeOnly mandatoryForBirthdayStart = new TimeOnly(21,0);

            if (TimeOnly.FromDateTime(birthdayDateTime) > mandatoryForBirthdayStart)

            {

                throw new ArgumentOutOfRangeException(nameof(birthdayDateTime),

                    "The mandatory for birthday "

                    + $"celebrating is at 21 o'clock. Curret time is {birthdayDateTime}");

            }

        }

        /// <exception cref="ArgumentOutOfRangeException"></exception>

        public static void ValidateEventTime(Event occasion)

        {

            TimeSpan minMeetingDuration = new TimeSpan(0,10,0);

            if (ActivityScheduler.GetActivitiesList().Any(activity =>

                {

                    TimeSpan timeBetweenEvents = Event.GetTimeBetweenEvents(occasion,activity);

                    if (timeBetweenEvents < minMeetingDuration)

                    {

                        return true;

                    }

                    return false;

                }))

            {

                throw new ArgumentOutOfRangeException(nameof(occasion),

                    "The time between events should be at least 10 minutes");

            }

        }

    }

}

using Labwork\_5.MainFlow;

namespace Labwork\_5

{

    public class ActivityScheduler

    {

        private static List<Event> s\_activities = new List<Event>();

        /// <exception cref="ArgumentOutOfRangeException"></exception>

        public static void AddActivity(Event activity)

        {

            Validator.ValidateEventTime(activity);

            s\_activities.Add(activity);

        }

        public static List<Event> GetActivitiesList()

        {

            return s\_activities;

        }

        public static Meeting GetLatestMeeting()

        {

            return s\_activities.Where(activity => activity is Meeting).MaxBy(activity => activity.DateTime) as Meeting;

        }

        public static void AssignDateToEvents(DateOnly date)

        {

            foreach (Event activity in ActivityScheduler.GetActivitiesList())

            {

                if (activity is Meeting meeting)

                {

                    meeting.DateTime = date.ToDateTime(TimeOnly.FromDateTime(meeting.DateTime));

                }

                else if (activity is Birthday birthday)

                {

                    birthday.DateTime = date.ToDateTime(TimeOnly.FromDateTime(birthday.DateTime));

                }

            }

        }

    }

}

using Labwork\_5.MainFlow;

namespace Labwork\_5

{

    public class Birthday : Event

    {

        public PersonModel Celebrant { get; set; } = new PersonModel();

        private string \_celebrationPlace;

        /// <exception cref="ArgumentException"></exception>

        public string CelebrationPlace

        {

            get => \_celebrationPlace;

            set

            {

                Validator.ValidatePlace(value);

                \_celebrationPlace = value;

            }

        }

        private int \_celebrantsAge;

        /// <exception cref="ArgumentOutOfRangeException"></exception>

        public int CelebrantsAge

        {

            get => \_celebrantsAge;

            set

            {

                Validator.ValidateAge(value);

                \_celebrantsAge = value;

            }

        }

        /// <exception cref="ArgumentException"></exception>

        /// <exception cref="ArgumentOutOfRangeException"></exception>

        public Birthday(DateTime dateTime, PersonModel celebrant, string celebrationPlace, int celebrantsAge) :

            base(dateTime)

        {

            Celebrant = celebrant;

            CelebrationPlace = celebrationPlace;

            CelebrantsAge = celebrantsAge;

        }

        public Birthday()

        {

        }

        public override string ToString()

        {

            return $"{base.ToString()}, Celebrant - {Celebrant}, CelebrationPlace - {CelebrationPlace},"

                + $" Celebrant's age - {CelebrantsAge}";

        }

    }

}

using Labwork\_5.MainFlow;

namespace Labwork\_5

{

    public abstract class Event

    {

        private DateTime \_dateTime;

        /// <exception cref="ArgumentOutOfRangeException"></exception>

        public DateTime DateTime

        {

            get => \_dateTime;

            set

            {

                if (this is Birthday)

                {

                    Validator.ValidateBirthdayTime(value);

                }

                else if (this is Meeting)

                {

                    Validator.ValidateMeetingTime(value);

                }

                \_dateTime = value;

            }

        }

        /// <exception cref="ArgumentOutOfRangeException"></exception>

        public Event(DateTime dateAndTime)

        {

            DateTime = new DateTime(dateAndTime.Year,dateAndTime.Month,dateAndTime.Day,

                dateAndTime.Hour, dateAndTime.Minute, dateAndTime.Second);

        }

        public Event()

        {

        }

        public static TimeSpan GetTimeBetweenEvents(Event leftSideEvent, Event rightSideEvent)

        {

            TimeSpan result = leftSideEvent.DateTime.Subtract(rightSideEvent.DateTime);

            if (result.Minutes < 0 || result.Hours < 0)

            {

                result.Negate();

            }

            return result;

        }

        public override string ToString()

        {

            return $"Date and Time - {DateTime.ToString()}";

        }

    }

}

using Labwork\_5.MainFlow;

namespace Labwork\_5

{

    public class Meeting : Event

    {

        public PersonModel PersonToMeet { get; set; } = new PersonModel();

        private string \_place;

        public string Place

        {

            get => \_place;

            set

            {

                Validator.ValidatePlace(value);

                \_place = value;

            }

        }

        public Meeting(DateTime dateAndTime, PersonModel person, string place) : base(dateAndTime)

        {

            PersonToMeet = person;

            Place = place;

        }

        public Meeting()

        {

        }

        public override string ToString()

        {

            return $"{base.ToString()}, Person to meet - {PersonToMeet}, Place - {Place}";

        }

    }

}

using Labwork\_5.MainFlow;

namespace Labwork\_5

{

    public class PersonModel

    {

        private string \_firstName;

        /// <exception cref="ArgumentException"></exception>

        public string FirstName

        {

            get => \_firstName;

            set

            {

                Validator.ValidateName(value);

                \_firstName = value;

            }

        }

        private string \_secondName;

        /// <exception cref="ArgumentException"></exception>

        public string SecondName

        {

            get => \_secondName;

            set

            {

                Validator.ValidateName(value);

                \_secondName = value;

            }

        }

        private string \_patronymic;

        /// <exception cref="ArgumentException"></exception>

        public string Patronymic

        {

            get => \_patronymic;

            set

            {

                Validator.ValidateName(value);

                \_patronymic = value;

            }

        }

        /// <exception cref="ArgumentException"></exception>

        public PersonModel(string firstName, string secondName, string patronymic)

        {

            FirstName = firstName;

            SecondName = secondName;

            Patronymic = patronymic;

        }

        public PersonModel()

        {

        }

        public override string ToString()

        {

            return $"First Name - {FirstName}, Second Name - {SecondName}, Patronymic - {Patronymic}";

        }

    }

}

**Python**

from datetime import \*

class DateTimeCapturer:

    @staticmethod

    def capture\_time():

        result = time()

        value\_is\_valid = False

        while not value\_is\_valid:

            value\_is\_valid = True

            try:

                result = time.fromisoformat(input("Please, enter the time: "))

            except ValueError as ve:

                print(ve)

                value\_is\_valid = False

        return result

    @staticmethod

    def capture\_date():

        result = date(1, 1, 1)

        value\_is\_valid = False

        while not value\_is\_valid:

            value\_is\_valid = True

            try:

                iso\_date = input("Please, enter the date: ")

                result = date.fromisoformat(iso\_date)

            except ValueError as ve:

                print(ve)

                value\_is\_valid = False

            except TypeError as te:

                print(te)

                value\_is\_valid = False

        return result

from datetime import \*

from activity\_scheduler import \*

from Capturer.datetime\_capturer import \*

from Capturer.person\_capturer import \*

from validator import \*

import functions

class EventCapturer:

    def \_\_init\_\_(self, meetings\_count):

        self.set\_meetings\_count(meetings\_count)

    def get\_meetings\_count(self):

        return self.\_\_meetings\_count

    def set\_meetings\_count(self, meetings\_count):

        Validator.validate\_meetings\_count(meetings\_count)

        self.\_\_meetings\_count = meetings\_count

    def capture\_events(self):

        while self.\_\_meetings\_count > 0:

            try:

                self.\_\_meetings\_count -= 1

                ActivityScheduler.add\_activity(EventCapturer.\_\_capture\_meeting())

                if self.\_\_meetings\_count == 0:

                    ActivityScheduler.add\_activity(self.\_\_capture\_birthday())

            except ValueError as ve:

                print(ve)

                self.\_\_meetings\_count += 1

        return ActivityScheduler.get\_activities\_list()

    @staticmethod

    def capture\_max\_meetings\_count():

        exception\_is\_thrown = True

        max\_meetings\_count = 0

        while exception\_is\_thrown:

            exception\_is\_thrown = False

            try:

                max\_meetings\_count = input("Please, enter maximal count of meetings per day: ")

            except TypeError as te:

                print(te)

                exception\_is\_thrown = True

            except ValueError as ve:

                print(ve)

                exception\_is\_thrown = True

        return max\_meetings\_count

    @staticmethod

    def \_\_capture\_place(activity):

        place = ""

        exception\_is\_thrown = True

        while exception\_is\_thrown:

            exception\_is\_thrown = False

            try:

                if activity is Birthday:

                    place = input("Please, enter the meeting place: ")

                    Birthday(activity).set\_celebration\_place(place)

                elif activity is Meeting:

                    place = input("Please, enter the celebration place: ")

                    Meeting(activity).set\_place(place)

            except ValueError as ve:

                print(ve)

                exception\_is\_thrown = True

        return place

    @staticmethod

    def \_\_capture\_meeting():

        meeting = Meeting(datetime(1, 1, 1), PersonModel(" ", " ", " "), " ")

        exception\_is\_thrown = True

        while exception\_is\_thrown:

            print("You are forming a meeting:")

            exception\_is\_thrown = False

            try:

                print("Please, identificate the person to meet:")

                PersonCapturer.capture\_person(meeting)

                EventCapturer.\_\_capture\_place(meeting)

                meeting.set\_date\_time(datetime.combine(datetime(meeting.get\_date\_time()).date(), DateTimeCapturer.capture\_time()))

                functions.print\_dash\_line()

            except ValueError as ve:

                print(ve)

                exception\_is\_thrown = True

        return meeting

    @staticmethod

    def \_\_capture\_birthday():

        birthday = Birthday(PersonModel(' ', ' ', ' '), datetime(1, 1, 1), 1, ' ')

        exception\_is\_thrown = True

        while exception\_is\_thrown:

            print("You are forming a birthday:")

            exception\_is\_thrown = False

            try:

                print("Please, identificate the celebrant:")

                PersonCapturer.capture\_person(birthday)

                EventCapturer.\_\_capture\_place(birthday)

                birthday.set\_date\_time(datetime.combine(datetime(birthday.get\_date\_time()).date(), DateTimeCapturer.capture\_time()))

                functions.print\_dash\_line()

            except ValueError as ve:

                print(ve)

                exception\_is\_thrown = True

        return birthday

from events import \*

from person\_model import \*

class PersonCapturer:

    @staticmethod

    def \_\_capture\_age(birthday):

        age = 0

        exception\_is\_thrown = True

        while exception\_is\_thrown:

            exception\_is\_thrown = False

            age = int(input("Please, enter the celebrant's age: "))

            try:

                Birthday(birthday).set\_age(age)

            except TypeError:

                print("The entered value isn't a number")

                exception\_is\_thrown = True

            except ValueError as ve:

                print(ve)

                exception\_is\_thrown = True

        return age

    @staticmethod

    def capture\_person(activity):

        person = PersonModel(' ', ' ', ' ')

        exception\_is\_thrown = True

        while exception\_is\_thrown:

            exception\_is\_thrown = False

            try:

                person.set\_first\_name(input("Please, enter the first name: "))

                person.set\_second\_name(input("Please, enter the second name: "))

                person.set\_patronymic(input("Please, enter the patronymic: "))

                if activity is Birthday:

                    PersonCapturer.\_\_capture\_age(activity)

                    Birthday(activity).set\_celebrant(person)

                elif activity is Meeting:

                    Meeting(activity).set\_person(person)

            except ValueError as ve:

                print(ve)

                exception\_is\_thrown = True

        return person

from validator import \*

from datetime import \*

from events import \*

class ActivityScheduler:

    \_\_activities = []

    @staticmethod

    def add\_activity(activity):

        Validator.validate\_event\_time(activity)

        ActivityScheduler.\_\_activities.append(activity)

    @staticmethod

    def get\_activities\_list():

        return ActivityScheduler.\_\_activities

    @staticmethod

    def get\_latest\_meeting():

        meetings = list(filter(lambda activity: activity is Meeting, ActivityScheduler.\_\_activities))

        max\_date\_time = max(list(map(lambda meeting: meeting.Meeting(meeting).get\_date\_time(), meetings)))

        return next(meeting for meeting in meetings if meeting.Meeting(meeting).get\_date\_time() == max\_date\_time())

    @staticmethod

    def assign\_date\_to\_events(date):

        for activity in ActivityScheduler.get\_activities\_list():

            if activity is Meeting:

                Meeting(activity).set\_date\_time(datetime.combine(date, datetime(Meeting(activity).get\_date\_time()).time()))

            elif activity is Birthday:

                Birthday(activity).set\_date\_time(datetime.combine(date, datetime(Birthday(activity).get\_date\_time()).time()))

from datetime import \*

from person\_model import \*

import validator

class Event:

    def \_\_init\_\_(self, date\_time = datetime(1, 1, 1)):

        self.set\_date\_time(date\_time)

    def get\_date\_time(self):

        return self.\_\_date\_time

    def set\_date\_time(self, date\_time):

        if self is Birthday:

            validator.Validator.validate\_birthday\_time(datetime(date\_time))

        elif self is Meeting:

            validator.Validator.validate\_meeting\_time(datetime(date\_time))

        self.\_\_date\_time = datetime(date\_time)

    def \_\_str\_\_(self) -> str:

        return self.get\_date\_time()

class Meeting(Event):

    def \_\_init\_\_(self, date\_time = datetime(1, 1, 1), person = PersonModel(), place = ' '):

        super().\_\_init\_\_(date\_time)

        self.set\_person(person)

        self.set\_place(place)

    def get\_place(self):

        return self.\_\_place

    def set\_place(self, place):

        validator.Validator.validate\_place(place)

        self.\_\_place = place

    def get\_person(self):

        return self.\_\_person

    def set\_person(self, person):

        self.\_\_person = person

    def \_\_str\_\_(self) -> str:

        return super().\_\_str\_\_() + self.get\_place() + self.get\_person()

class Birthday(Event):

    def \_\_init\_\_(self, celebrant = PersonModel(), date\_time = datetime(1, 1, 1), age = 1, celebration\_place = ' '):

        super().\_\_init\_\_(date\_time)

        self.set\_celebrant(celebrant)

        self.set\_age(age)

        self.set\_celebration\_place(celebration\_place)

    def get\_age(self):

        return self.\_\_age

    def set\_age(self, age):

        validator.Validator.validate\_age(age)

        self.\_\_age = age

    def get\_celebration\_place(self):

        return self.\_\_celebration\_place

    def set\_celebration\_place(self, celebration\_place):

        validator.Validator.validate\_place(celebration\_place)

        self.\_\_celebration\_place = celebration\_place

    def get\_celebrant(self):

        return self.\_\_celebrant

    def set\_celebrant(self, celebrant):

        self.\_\_celebrant = celebrant

    def \_\_str\_\_(self) -> str:

        return super().\_\_str\_\_() + self.\_\_place

from datetime import timedelta

from events import \*

from activity\_scheduler import \*

def parse\_duration(duration):

    seconds = abs(timedelta(duration))

    duration = timedelta(seconds=seconds)

def print\_events(events):

    for event in events:

        print(event)

def print\_dash\_line():

    print('-' \* 70)

def get\_birthday\_from\_list(activities):

    return next(activity for activity in activities if Event(activity) is Birthday)

def get\_time\_between\_events(first\_event, second\_event):

        result = timedelta(Event(first\_event).get\_date\_time() - Event(second\_event).get\_date\_time())

        if result.seconds // 3600 < 0 or (result.seconds // 60) % 60 < 0:

            result = timedelta(abs(result.total\_seconds()))

        return result

def validate\_event\_time(activity):

        min\_meeting\_duration = timedelta(minutes=10)

        for activity\_from\_list in ActivityScheduler.get\_activities\_list():

            time\_between\_events = timedelta(abs(timedelta(functions.get\_time\_between\_events(activity, activity\_from\_list)).total\_seconds()))

            if time\_between\_events < min\_meeting\_duration:

                raise ValueError("The time between events should be at least 10 minutes")

from datetime import datetime

from Capturer.event\_capturer import \*

from Capturer.datetime\_capturer import \*

from person\_model import \*

from activity\_scheduler import \*

from events import \*

import functions

print("Please, enter the concrete date when you are free from chores: ")

concrete\_date = DateTimeCapturer.capture\_date()

event\_capturer = EventCapturer(int(EventCapturer.capture\_max\_meetings\_count()))

functions.print\_dash\_line()

activities = event\_capturer.capture\_events()

ActivityScheduler.assign\_date\_to\_events(concrete\_date)

functions.print\_events()

print("The last meeting of the day:")

last\_meeting = ActivityScheduler.get\_latest\_meeting()

print(last\_meeting)

functions.print\_dash\_line()

birthday = functions.get\_birthday\_from\_list()

print("Period of time between last meeting and birthday: "

    + Event.get\_time\_between\_events(birthday, last\_meeting))

from validator import \*

class PersonModel:

    def \_\_init\_\_(self, first\_name = ' ', second\_name = ' ', patronymic = ' '):

        self.set\_first\_name(first\_name)

        self.set\_second\_name(second\_name)

        self.set\_patronymic(patronymic)

    def get\_first\_name(self):

        return self.\_\_first\_name

    def set\_first\_name(self, first\_name):

        Validator.validate\_name(first\_name)

        self.\_\_first\_name = first\_name

    def get\_second\_name(self):

        return self.\_\_second\_name

    def set\_second\_name(self, second\_name):

        Validator.validate\_name(second\_name)

        self.\_\_second\_name = second\_name

    def get\_patronymic(self):

        return self.\_\_patronymic

    def set\_patronymic(self, patronymic):

        Validator.validate\_name(patronymic)

        self.\_\_patronymic = patronymic

    def \_\_str\_\_(self) -> str:

        return f"First name - {self.get\_first\_name()}, Second name - {self.get\_second\_name()}, Patronymic - {self.get\_patronymic()}"

from datetime import \*

class Validator:

    @staticmethod

    def validate\_name(name):

        if name == "":

            raise ValueError("Name musn't be empty")

    @staticmethod

    def validate\_age(age):

        if age < 1:

            raise ValueError("The entered age musn't be less than 1")

    @staticmethod

    def validate\_meetings\_count(meetings\_count):

        if meetings\_count < 0:

            raise ValueError("The count of meetings per day can't be less than 0")

        max\_meetings\_count = 126

        if meetings\_count > max\_meetings\_count:

            raise ValueError("The count of meetings per day can't be "

                + "more than 126 (at least 10 minutes per meeting and 3 hours for birthday)")

    @staticmethod

    def validate\_place(place):

        if str(place) == "":

            raise ValueError("The name of place shouldn't be empty")

    @staticmethod

    def validate\_meeting\_time(meeting\_date\_time):

        last\_available\_meeting\_time = time(20, 50)

        if datetime(meeting\_date\_time).time() > last\_available\_meeting\_time:

            raise ValueError("The time of event shouldn't be bigger than last available (20:50)")

    @staticmethod

    def validate\_birthday\_time(birthday\_date\_time):

        mandatory\_for\_birthday\_start = time(21, 0, 0)

        if datetime(birthday\_date\_time).time() > mandatory\_for\_birthday\_start:

            raise ValueError("The mandatory for birthday celebrating is at 21 o'clock."

                + f" Current time is {birthday\_date\_time}")

**Висновок**

На цій лабораторній роботі я засвоїв матеріал про поліморфізм та наслідування об’єктів класів. Мені довелося створити декількох нащадків базового абстрактного класу, та використовувати властивість про те, що вони «є» їх предком.