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) *ІП-15 Плугатирьов Дмитро Валерійович*
(шифр, прізвище, ім'я, по батькові)

Перевірив *Вечерковська Анастасія Сергіївна*
(прізвище, ім'я, по батькові)

Київ 2022

УСПАДКУВАННЯ ТА ПОЛІМОРФІЗМ

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

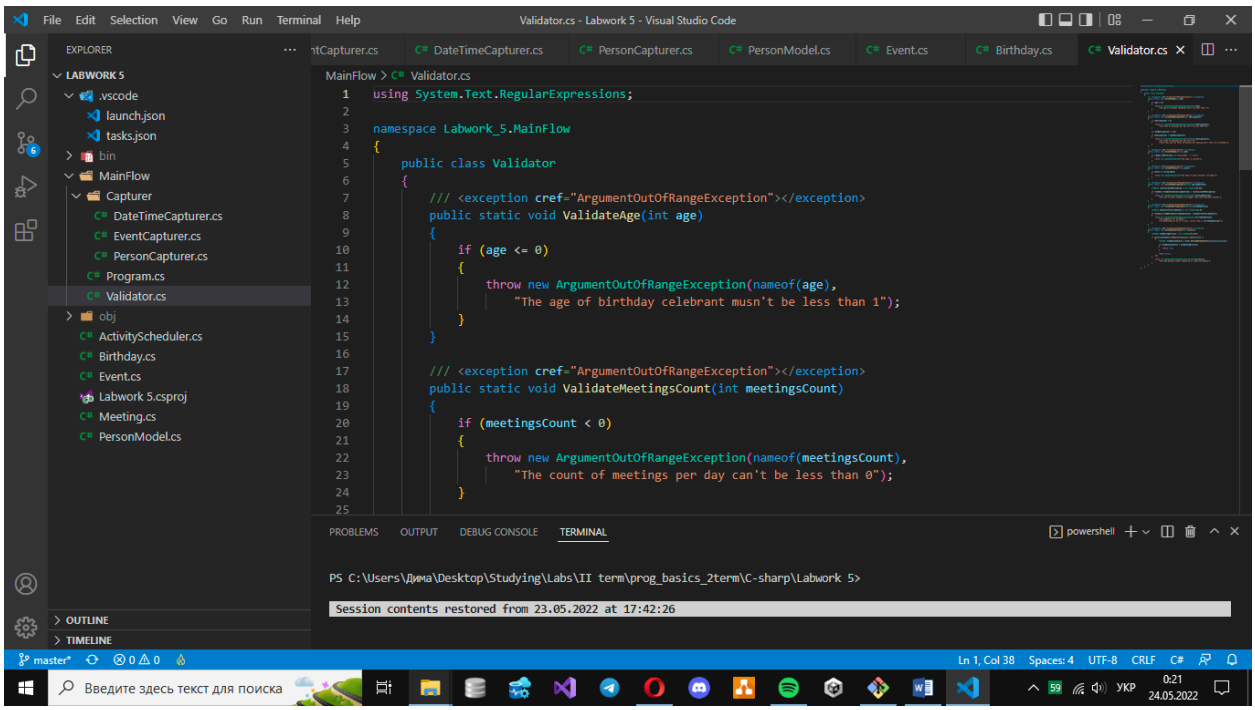
Варіант 25

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

C#




```
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_dat
e_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.