

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**Лабораторна робота №** **5**

з дисципліни “Основи програмування”

тема “Формат даних XML”

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | Виконав  студент I курсу | | групи КП-01 | | Пецеля Артем Володимирович | | (*прізвище, ім’я, по батькові*) |   варіант №12 |  | Перевірив  “\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 20\_\_\_ р.  викладач  Гадиняк Руслан Анатолійович  (*прізвище, ім’я, по батькові*) |

Київ 2021

**Мета роботи**

Навчитись виконувати серіалізацію і десереалізацію даних у форматі XML.

Виконати генерацію зображення з графіком на основі вхідних даних.

**Постановка завдання**

Задано файл з даними у форматі XML.

Створити консольну програму, що дозволяє користувачу виконувати операції над даними із файлів у форматі XML заданої структури: десеріалізувати набір даних із файлу, згенерувати і зберегти частину даних у новий XML файл, обчислити і вивести дані за варіантом, а також вивести задані дані на зображення з графіком та зберегти його у файл.

Користувач керує програмою за допомогою командного інтерфейсу у консолі.

Консольні команди користувача:

* **load {filename}** - десеріалізувати XML із заданого файлу у об’єкти в процесі.
* **print {pageNum}** - вивести загальну кількість сторінок і дані сторінки (за номером) десеріалізованих даних з об’єктів у консоль. Розмір сторінки довільний.
* **save {filename}** - серіалізувати всі дані у заданий XML файл (з відступами).
* **export {N} {filename}** - серіалізувати частину даних у XML файл за варіантом (з відступами).
* **subjects -** Список назв всіх унікальних навчальних предметів (subj)
* **subject {subj} -** Список назв курсів (title) обраного предмету (subj)
* **instructors -** Список всіх унікальних викладачів (instructor)
* **image {filename} -** створити і зберегти зображення з графіком сумарної кількості кредитів (units) по кожному предмету (subj).

**Текст коду програми**

|  |
| --- |
| **Program.cs** |
| using System;  namespace lab5  {      class Program      {          static void Main(string[] args)          {              ConsoleInterface.Run();          }      }  } |

|  |
| --- |
| **Course.cs** |
| using System;  using System.Xml.Serialization;  namespace lab5  {      public class Course      {          [XmlElement("reg\_num")]          public int registrationNumber;          [XmlElement("subj")]          public string subject;          [XmlElement("crse")]          public int courceId;          [XmlElement("sect")]          public string section;          public string title;          public float units;          public string instructor;          public string days;          public Time time;          public Place place;          public override string ToString()          {              return $"[{registrationNumber}]  {courceId}) {title}. Subject: {subject}. - {section} - Units: {units}. Instructor: {instructor}. At {days} - {time} - {place}";          }      }  } |

|  |
| --- |
| **Time.cs** |
| using System;  using System.Xml.Serialization;  namespace lab5  {      public class Time      {          [XmlElement("start\_time")]          public string startTime;          [XmlElement("end\_time")]          public string endTime;          public override string ToString()          {              return $"{startTime} till {endTime}";          }      }  } |

|  |
| --- |
| **Place.cs** |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace lab5  {      public class Place      {          public string building;          public string room;          public override string ToString()          {              return $"{building}, room {room}";          }      }  } |

|  |
| --- |
| **Root.cs** |
| using System;  using System.Collections.Generic;  using System.Xml.Serialization;  namespace lab5  {      [XmlType("root")]      public class Root      {          [XmlElement("course")]          public List<Course> courses;      }  } |

|  |
| --- |
| **ConsoleInterfacecs** |
| using System;  using static System.Console;  using System.Collections.Generic;  namespace lab5  {      static class ConsoleInterface      {          public static void Run()          {              bool exit = false;              DataProcessor dataProcessor = new DataProcessor();              while (!exit)              {                  string command = ReadCommand();                  Arguments args;                  try                  {                      args = ParseCommand(command);                  }                  catch (Exception ex)                  {                      WriteLine($"Parsing error: {ex.Message}");                      continue;                  }                  try                  {                      switch (args.operation)                      {                          case "load":                              {                                  ProcessLoad(args, dataProcessor);                                  break;                              }                          case "print":                              {                                  ProcessPrint(args, dataProcessor);                                  break;                              }                          case "save":                              {                                  ProcessSave(args, dataProcessor);                                  break;                              }                          case "export":                              {                                  ProcessExport(args, dataProcessor);                                  break;                              }                          case "subjects":                              {                                  ProcessSubjects(args, dataProcessor);                                  break;                              }                          case "subject":                              {                                  ProcessSubject(args, dataProcessor);                                  break;                              }                          case "instructors":                              {                                  ProcessInstructors(args, dataProcessor);                                  break;                              }                          case "image":                              {                                  ProcessImage(args, dataProcessor);                                  break;                              }                          case "help":                              {                                  WriteLine(GetHelp());                                  break;                              }                          case "exit":                          case "":                              {                                  exit = true;                                  break;                              }                      }                  }                  catch (Exception ex)                  {                      WriteLine($"Processing error: {ex.Message}");                  }              }          }          private static void ProcessImage(Arguments args, DataProcessor dataProcessor)          {              if (args.otherArguments.Length != 1)              {                  throw new ArgumentException($"Operation 'image' should have 1 argument. Got: {args.otherArguments.Length}");              }              if (dataProcessor.Courses.Count == 0)              {                  throw new Exception("Buffer does not contains any cources to generate graphics");              }              string[] labels = dataProcessor.GetUniqueSubjects();              float[] values = dataProcessor.GetSums();              GraphicsGenerator.CreateGraphics(args.otherArguments[0], labels, values);              WriteLine($"Image was saved to {args.otherArguments[0]}");          }          private static void ProcessSubjects(Arguments args, DataProcessor dataProcessor)          {              if (args.otherArguments.Length != 0)              {                  throw new ArgumentException($"Operation 'subjects' should have 0 argument. Got: {args.otherArguments.Length}");              }              string[] subjects = dataProcessor.GetUniqueSubjects();              if (subjects.Length != 0)              {                  WriteLine("Subjects:");                  for (int i = 0; i < subjects.Length; i++)                  {                      WriteLine(subjects[i]);                  }              }              else              {                  WriteLine("There is no subjects");              }          }          private static void ProcessSubject(Arguments args, DataProcessor dataProcessor)          {              if (args.otherArguments.Length != 1)              {                  throw new ArgumentException($"Operation 'subject' should have 0 argument. Got: {args.otherArguments.Length}");              }              string[] titles = dataProcessor.GetTitlesBySubject(args.otherArguments[0]);              if (titles.Length != 0)              {                  WriteLine("Courses:");                  for (int i = 0; i < titles.Length; i++)                  {                      WriteLine(titles[i]);                  }              }              else              {                  WriteLine($"There is no courses for this subject: {args.otherArguments[0]}");              }          }          private static void ProcessInstructors(Arguments args, DataProcessor dataProcessor)          {              if (args.otherArguments.Length != 0)              {                  throw new ArgumentException($"Operation 'instructors' should have 0 argument. Got: {args.otherArguments.Length}");              }              string[] instructors = dataProcessor.GetUniqueInstructors();              if (instructors.Length != 0)              {                  WriteLine("Instructors:");                  for (int i = 0; i < instructors.Length; i++)                  {                      WriteLine(instructors[i]);                  }              }              else              {                  WriteLine("There is no instructors");              }          }          private static void ProcessExport(Arguments args, DataProcessor dataProcessor)          {              if (args.otherArguments.Length != 2)              {                  throw new ArgumentException($"Operation 'export' should have 2 argument. Got: {args.otherArguments.Length}");              }              if (!int.TryParse(args.otherArguments[0], out int n) && n > 0)              {                  throw new ArgumentException($"Value should be positive integer. Got: {args.otherArguments[0]}");              }              List<Course> exportData = dataProcessor.GetExport(n);              XmlDataIO.StoreCoursesToFile(args.otherArguments[1], exportData);              WriteLine($"{exportData.Count} courses was exported to {args.otherArguments[1]}");          }          private static void ProcessSave(Arguments args, DataProcessor dataProcessor)          {              if (args.otherArguments.Length != 1)              {                  throw new ArgumentException($"Operation 'save' should have 1 argument. Got: {args.otherArguments.Length}");              }              XmlDataIO.StoreCoursesToFile(args.otherArguments[0], dataProcessor.Courses);          }          private static void ProcessPrint(Arguments args, DataProcessor dataProcessor)          {              if (args.otherArguments.Length != 1)              {                  throw new ArgumentException($"Operation 'print' should have 1 argument. Got: {args.otherArguments.Length}");              }              if (!int.TryParse(args.otherArguments[0], out int pageNumber))              {                  throw new ArgumentException($"Value should be integer. Got: {args.otherArguments[0]}");              }              List<Course> page = dataProcessor.GetPage(pageNumber);              WriteLine($"Page: {pageNumber}/{dataProcessor.GetPageCount()}\n");              foreach(Course course in page)              {                  WriteLine(course);              }          }          private static void ProcessLoad(Arguments args, DataProcessor dataProcessor)          {              if (args.otherArguments.Length != 1)              {                  throw new ArgumentException($"Operation 'load' should have 1 argument. Got: {args.otherArguments.Length}");              }              List<Course> courses = XmlDataIO.GetCoursesFromFile(args.otherArguments[0]);              dataProcessor.Courses = courses;              WriteLine($"{courses.Count} courses was loaded from {args.operation[0]}");          }          private static string GetHelp()          {              string[] commands = new string[] { "load {filePath}", "print {pageNum}", "save {filePath}",                  "export {N} {filePath}", "subjects", "subject {subj}", "instructors" , "image {filePath}" };              string[] descriptions = new string[] { "deserialize data from XML file", "print page of data", "serialize all data XML to XML file",                  "serialize part of data to XML file", "list of all unique subjects", "list of cources of subject", "list of all unique instructors" , "create graphics" };              string helpString = "";              for (int i = 0; i < commands.Length; i++)              {                  helpString += commands[i] + " - " + descriptions[i] + (i == commands.Length - 1 ? "" : "\n");              }              return helpString;          }          private static Arguments ParseCommand(string command)          {              string[] subcommads = command.Trim().Split(" ");              ValidateCommandLength(subcommads.Length);              string operation = subcommads[0];              ValidateOperations(operation);              string[] otherArguments = new string[subcommads.Length - 1];              for (int i = 0; i < otherArguments.Length; i++)              {                  otherArguments[i] = subcommads[i + 1];              }              return new Arguments()              {                  operation = operation,                  otherArguments = otherArguments              };          }          private static void ValidateOperations(string operation)          {              string[] validOperations = new string[] { "load", "print", "save", "export", "subjects", "subject", "instructors", "image", "help", "exit", "" };              for (int i = 0; i < validOperations.Length; i++)              {                  if (validOperations[i] == operation)                  {                      return;                  }              }              throw new ArgumentException($"Unknown command");          }          private static void ValidateCommandLength(int length)          {              if (length < 1)              {                  throw new ArgumentException($"Command length should be more than 0. Got: {length}");              }          }          struct Arguments          {              public string operation;              public string[] otherArguments;          }          private static string ReadCommand()          {              Write("> ");              return ReadLine();          }      }  } |

|  |
| --- |
| **DataProcessor.cs** |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace lab5  {      class DataProcessor      {          private List<Course> courses;          private HashSet<string> instructors;          private Dictionary<string, List<Course>> subjectDictionary;          private HashSet<string> subjects;          private Dictionary<float, List<Course>> unitDictionary;          private SortedSet<float> units;          private Dictionary<string, float> sumDictionary;          private const int pageSize = 10;          public DataProcessor()          {              this.courses = new List<Course>();              Initialize();          }          private void Initialize()          {              this.instructors = new HashSet<string>();              this.subjectDictionary = new Dictionary<string, List<Course>>();              this.subjects = new HashSet<string>();              this.unitDictionary = new Dictionary<float, List<Course>>();              this.units = new SortedSet<float>();              this.sumDictionary = new Dictionary<string, float>();          }          public List<Course> Courses          {              get              {                  return courses;              }              set              {                  this.courses = value;                  Initialize();                  foreach (Course course in this.courses)                  {                      if (course.instructor != null)                      {                          instructors.Add(course.instructor);                      }                      if (course.subject != null)                      {                          subjects.Add(course.subject);                          if (subjectDictionary.TryGetValue(course.subject, out List<Course> list1))                          {                              list1.Add(course);                              subjectDictionary.Remove(course.subject);                              subjectDictionary.Add(course.subject, list1);                          }                          else                          {                              List<Course> newList1 = new List<Course>();                              newList1.Add(course);                              subjectDictionary.Add(course.subject, newList1);                          }                      }                      units.Add(course.units);                      if (unitDictionary.TryGetValue(course.units, out List<Course> list2))                      {                          list2.Add(course);                          unitDictionary.Remove(course.units);                          unitDictionary.Add(course.units, list2);                      }                      else                      {                          List<Course> newList2 = new List<Course>();                          newList2.Add(course);                          unitDictionary.Add(course.units, newList2);                      }                      float sum = sumDictionary.GetValueOrDefault(course.subject);                      sum += course.units;                      sumDictionary.Remove(course.subject);                      sumDictionary.Add(course.subject, sum);                  }              }          }          public List<Course> GetPage(int n)          {              int pageCount = GetPageCount();              if (n < 1 || n > pageCount)              {                  throw new Exception("Wrong page");              }              List<Course> page = new List<Course>();              for (int i = (n-1) \* pageSize; i < Math.Min(n \* pageSize, courses.Count); i++)              {                  page.Add(courses[i]);              }              return page;          }          public int GetPageCount()          {              return (int)Math.Ceiling(courses.Count / (double)pageSize);          }          public List<Course> GetExport(int n)          {              List<Course> export = new List<Course>();              float[] unitsArray = new float[units.Count];              units.CopyTo(unitsArray);              for (int i = unitsArray.Length - 1; i >= 0; i--)              {                  unitDictionary.TryGetValue(unitsArray[i], out List<Course> listPerUnit);                  for (int j = 0; j < listPerUnit.Count; j++)                  {                      if (export.Count >= n)                      {                          return export;                      }                      export.Add(listPerUnit[j]);                  }              }              return export;          }          public string[] GetUniqueSubjects()          {              string[] subjects = new string[this.subjects.Count];              this.subjects.CopyTo(subjects);              return subjects;          }          public string[] GetTitlesBySubject(string subject)          {              if (subjectDictionary.TryGetValue(subject, out List<Course> list))              {                  string[] titles = new string[list.Count];                  for (int i = 0; i < list.Count; i++)                  {                      titles[i] = list[i].title;                  }                  return titles;              }              return new string[0];          }          public string[] GetUniqueInstructors()          {              string[] instructors = new string[this.instructors.Count];              this.instructors.CopyTo(instructors);              return instructors;          }          public float[] GetSums()          {              string[] subjects = GetUniqueSubjects();              float[] sums = new float[subjects.Length];              for (int i = 0; i < subjects.Length; i++)              {                  sums[i] = sumDictionary.GetValueOrDefault<string, float>(subjects[i]);              }              return sums;          }      }  } |

|  |
| --- |
| **GraphicsGenerator.cs** |
| using System;  using ScottPlot;  namespace lab5  {      static class GraphicsGenerator      {          public static void CreateGraphics(string filePath, string[] labels, float[] values)          {              Plot plot = new Plot(600, 400);              double[] ys = ConvertToDoubleArray(values);              double[] xs = new double[labels.Length];              FillNatural(xs);              plot.PlotBar(xs, ys, horizontal: true);              plot.YTicks(xs, labels);              plot.SaveFig(filePath);          }          private static void FillNatural(double[] array)          {              for (int i = 0; i < array.Length; i++)              {                  array[i] = i + 1;              }          }          private static double[] ConvertToDoubleArray(float[] array)          {              double[] doubleArray = new double[array.Length];              for (int i = 0; i < array.Length; i++)              {                  doubleArray[i] = array[i];              }              return doubleArray;          }      }  } |

|  |
| --- |
| **XmlDataIO.cs** |
| using System;  using System.Collections.Generic;  using System.Xml.Serialization;  using System.IO;  namespace lab5  {      static class XmlDataIO      {          public static void StoreCoursesToFile(string filePath, List<Course> courses)          {              XmlSerializer ser = new XmlSerializer(typeof(Root));              StreamWriter writer = new StreamWriter(filePath);              Root root = new Root();              root.courses = courses;              ser.Serialize(writer, root);              writer.Close();          }          public static List<Course> GetCoursesFromFile(string filePath)          {              ValidateFile(filePath);              XmlSerializer ser = new XmlSerializer(typeof(Root));              StreamReader reader = new StreamReader(filePath);              Root root;              try              {                  root = (Root)ser.Deserialize(reader);              }              catch              {                  throw new Exception("Cannot deserialize file");              }              reader.Close();              return root.courses;          }          private static void ValidateFile(string filePath)          {              if (!File.Exists(filePath))              {                  throw new ArgumentException("File does not exist");              }          }      }  } |

**Вхідне DOM дерево**

|  |
| --- |
| **reed.xml** |
| <root>    <course>      <reg\_num>10577</reg\_num>      <subj>ANTH</subj>      <crse>211</crse>      <sect>F01</sect>      <title>Introduction to Anthropology</title>      <units>1.0</units>      <instructor>Brightman</instructor>      <days>M-W</days>      <time>        <start\_time>03:10PM</start\_time>        <end\_time>04:30</end\_time>      </time>      <place>        <building>ELIOT</building>        <room>414</room>      </place>    </course>    <course>      <reg\_num>20573</reg\_num>      <subj>ANTH</subj>      <crse>344</crse>      <sect>S01</sect>      <title>Sex and Gender</title>      <units>1.0</units>      <instructor>Makley</instructor>      <days>T-Th</days>      <time>        <start\_time>10:30AM</start\_time>        <end\_time>11:50</end\_time>      </time>      <place>        <building>VOLLUM</building>        <room>120</room>      </place>    </course>  </root> |

|  |
| --- |
| **Графічне представлення DOM дерева reed.xml** |
|  |

**Приклади результатів**

**Приклад 1.** Команди load та print

|  |
| --- |
| > load ./reed.xml  703 courses was loaded from ./reed.xml  > print 1  Page: 1/71  [10577] 211) Introduction to Anthropology. Subject: ANTH. - F01 - Units: 1. Instructor: Brightman. At M-W - 03:10PM till 04:30 - ELIOT, room 414  [20573] 344) Sex and Gender. Subject: ANTH. - S01 - Units: 1. Instructor: Makley. At T-Th - 10:30AM till 11:50 - VOLLUM, room 120  [10624] 431) Field Biology of Amphibians. Subject: BIOL. - F01 - Units: 0,5. Instructor: Kaplan. At T - 06:10PM till 08:00 - PHYSIC, room 240A  [10626] 431) Bacterial Pathogenesis. Subject: BIOL. - F03 - Units: 0,5. Instructor: . At - till - , room Mellies RESCHEDULED TO OTHER SEMESTER  [20626] 431) Seminar in Biology. Subject: BIOL. - S04 - Units: 0,5. Instructor: Yezerinac. At Th - 06:10PM till 08:00 - BIOL, room 200A  [10543] 101) MolecularStructure and Properties. Subject: CHEM. - F - Units: 1. Instructor: Geselbracht. At M-W-F - 11:00AM till 11:50 - VOLLUM, room VLH  [10544] 101) MolecularStructure and Properties. Subject: CHEM. - F01 - Units: 0. Instructor: Geselbracht. At M - 01:10PM till 02:00 - CHEM, room 301  [10545] 101) MolecularStructure and Properties. Subject: CHEM. - F02 - Units: 0. Instructor: Geselbracht. At M - 02:10PM till 03:00 - CHEM, room 301  [10546] 101) MolecularStructure and Properties. Subject: CHEM. - F03 - Units: 0. Instructor: Geselbracht. At M - 03:10PM till 04:00 - CHEM, room 301  [10789] 101) MolecularStructure and Properties. Subject: CHEM. - F08 - Units: 0. Instructor: Geselbracht. At T - 11:00AM till 11:50 - PHYSIC, room 123 |

**Приклад 2.** Команда print з неправильним аргументом

|  |
| --- |
| > print 72  Processing error: Wrong page |

**Приклад 3.** Команда export

|  |
| --- |
| > export 5 ./out.xml  5 courses was exported to ./out.xml |

|  |
| --- |
| **out.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">    <course>      <reg\_num>10577</reg\_num>      <subj>ANTH</subj>      <crse>211</crse>      <sect>F01</sect>      <title>Introduction to Anthropology</title>      <units>1</units>      <instructor>Brightman</instructor>      <days>M-W</days>      <time>        <start\_time>03:10PM</start\_time>        <end\_time>04:30</end\_time>      </time>      <place>        <building>ELIOT</building>        <room>414</room>      </place>    </course>    <course>      <reg\_num>20573</reg\_num>      <subj>ANTH</subj>      <crse>344</crse>      <sect>S01</sect>      <title>Sex and Gender</title>      <units>1</units>      <instructor>Makley</instructor>      <days>T-Th</days>      <time>        <start\_time>10:30AM</start\_time>        <end\_time>11:50</end\_time>      </time>      <place>        <building>VOLLUM</building>        <room>120</room>      </place>    </course>    <course>      <reg\_num>10543</reg\_num>      <subj>CHEM</subj>      <crse>101</crse>      <sect>F</sect>      <title>MolecularStructure and Properties</title>      <units>1</units>      <instructor>Geselbracht</instructor>      <days>M-W-F</days>      <time>        <start\_time>11:00AM</start\_time>        <end\_time>11:50</end\_time>      </time>      <place>        <building>VOLLUM</building>        <room>VLH</room>      </place>    </course>    <course>      <reg\_num>20575</reg\_num>      <subj>ANTH</subj>      <crse>344</crse>      <sect>S02</sect>      <title>Sex and Gender</title>      <units>1</units>      <instructor>Makley</instructor>      <days>T-Th</days>      <time>        <start\_time>01:10PM</start\_time>        <end\_time>02:30</end\_time>      </time>      <place>        <building>VOLLUM</building>        <room>110</room>      </place>    </course>  </root> |

|  |
| --- |
| **Графічне представлення DOM дерева out.xml** |
|  |

**Приклад 4.** Команда subject

|  |
| --- |
| > subject BIOL  Courses:  Field Biology of Amphibians  Bacterial Pathogenesis  Seminar in Biology  Intro Biology Lect and Lab  Intro Biology Lect and Lab  Intro Biology Lect and Lab  Intro Biology Lect and Lab  Intro Biology Lect and Lab  Intro Biology Lect and Lab  Intro Biology Lect and Lab  Intro Biology Lect and Lab  Intro Biology Lect and Lab  Intro Biology Lect and Lab  Intro Biology Lect and Lab  Intro Biology Lect and Lab  Intro Biology Lect and Lab  Intro Biology Lect and Lab  Intro Biology Lect and Lab  Intro Biology Lect and Lab  Intro Biology Lect and Lab  Intro Biology Lect and Lab  Vascular Plant Diversity  Vascular Plant Diversity  Vascular Plant Diversity  Animal Behav. and Behav.Ecology  Animal Behav. and Behav.Ecology  Animal Behav. and Behav.Ecology  Developmental Biology  Developmental Biology  Developmental Biology  Genetics and Gene Regulation  Genetics and Gene Regulation  Genetics and Gene Regulation  Microbiology  Microbiology  Microbiology  Microbiology  Microbiology  Microbiology  Genetics and MolecularBiology  Genetics and MolecularBiology  Genetics and MolecularBiology  Genetics and MolecularBiology  Population Bio:Ecolgy and Evolutn  Population Bio:Ecolgy and Evolutn  Population Bio:Ecolgy and Evolutn  Scientific Computation  Cellular Biology  Cellular Biology  Cellular Biology  Animal Physiology  Animal Physiology  Animal Physiology  Evolutionary Biology  Evolutionary Biology  Evolutionary Biology |

**Приклад 5.** Генерація графіку

|  |
| --- |
| > image ./exp.png  Image was saved to ./exp.png |

Chart, bar chart

Description automatically generated

**Висновки**

Виконавши дану лабораторну роботу було реалізовано модуль серіалізації та десеріалізації даних у форматі XML. Для обробки отриманих даних були застосовані різні структури даних із стандартної бібліотеки C#, такі як Dictionary, HashSet, SortedSet.