Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ» (МОСКОВСКИЙ ПОЛИТЕХ)

КАФЕДРА СМАРТ-ТЕХНОЛОГИИ

ОТЧЁТ ПО УЧЕБНОЙ ПРАКТИКЕ

По дисциплине **«**Инженерное проектирование**»**
по направлению 09.03.01 Информатика и вычислительная техника
Образовательная программа (профиль) «Разработка инженерного программного обеспечения»

Преподаватель: / / Сивцев Алексей Олегович / *подпись ФИО*

Студент: / / Балынин Егор Дмитриевич, 241-3210 / *подпись ФИО, группа*

Москва, 2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования «МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» ЗАДАНИЕ НА УЧЕБНУЮ (ПРОЕКТНУЮ) ПРАКТИКУ

по направлению 09.03.01 Информатика и вычислительная техника

Образовательная программа (профиль) «Разработка и интеграция инженерного ПО»

Задачи

1. Ознакомление с деятельностью компании «Нанософт-разработка», структура компании, основные задачи, технологии. Совместные проекты с Московским Политехом.
2. Обзор предметной области. Изучение САПР nanoCAD, дополнительная практика программирования на C++ и C#.
3. Прохождение курса «Использование .NET API nanoCAD»
    - Модуль 1. Введение в .NET API. Ввод данных, ключевые слова. Транзакции, создание объектов. Выбор, наборы выбора, приведение типов.
    - Модуль 2. Сложные составные объекты. Блоки, полилинии, размеры.
    - Модуль 3. Геометрическая библиотека, работа с кривыми. Координатные системы, команды в ПСК.
    - Модуль 4. Основные свойства примитивов. Интерактивное создание и редактирование примитивов: Jig.
    - Модуль 5. Реакторы. События базы данных, редактора, системы. Контроль объектов, находящихся под курсором.
    - Модуль 6. Создание окон на WinForms/WPF. Палитры. Программный интерфейс подсистемы печати.
    - Модуль 7. Таблицы .dwg. Неграфические данные. XData, XRecord, словари расширения. Переопределение характеристик объектов чертежа: Overrule.

- Модуль 8. MultiCAD.NET. Элементы оформления и таблицы.

4. Изучение работы JavaScript в браузере
    - Разбор основ работы JS в браузере: DOM, события, асинхронность.
    - Практическое применение JS в разработке веб-страниц.
5. Освоение фреймворка React
    - Основы React: компоненты, состояние, хуки.
    - Создание первых приложений на React.
    - Управление состоянием и маршрутизация.
6. Изучение Figma для проектирования интерфейсов
    - Создание макетов с нуля.
    - Работа с сетками, компонентами, шрифтами и цветами.
    - Подготовка дизайна для адаптивной верстки.
7. Проектирование макетов сайта в Figma
    - Создание макетов для основной страницы, страницы команд и страницы проектов.
    - Совместная работа с другим фронтенд-разработчиком.
    - Оптимизация UI/UX.
8. Верстка страниц на React
    - Адаптивная верстка с использованием HTML5 и CSS.
    - Интеграция макетов в код.
    - Разработка компонентов и логики для отображения данных.
9. Доработка и тестирование проекта
    - Проверка корректного отображения на разных устройствах.
    - Оптимизация кода и исправление багов.
    - Улучшение дизайна и пользовательского опыта.
10. Разработка программного модуля с использованием API nanoCAD.

Роль в команде, участие в командной работе

Роль: Разработчик САПР, веб-разработчик

Мероприятия: участие в еженедельных собраниях, подготовка к промежуточным аттестациям, разработка отчетной документации, хакатон(конференции, выставки), организация командной работы

РУКОВОДИТЕЛЬ:
«___» _____ 2025, _____ / Сивцев А. О.
подпись ФИО
СТУДЕНТ:
«___» _____ 2025, _____ / Балынин Е.Д. , 241-3210
подпись ФИО, группа

ДНЕВНИК УЧЕБНОЙ ПРАКТИКИ

Даты проведения: 03.02.2025-26.05.2025

Студент: Балынин Егор Дмитриевич Группа: 241-3210

Место прохождения практики: Московский Политехнический Университет

Руководитель практики от образовательной организации: Сивцев А. О.

Руководитель практики от профильной организации: Сивцев А. О.

Инструктаж по технике
безопасности провел Сивцев Алексей Олегович

фио дата подпись

Инструктаж по технике
безопасности провел Сивцев Алексей Олегович

фио дата подпись

С техникой безопасности
ознакомлен: Балынин Егор Дмитриевич

фио дата подпись

Даты	**Виды и основное содержание работы**	**Отметка о выполнении работы руководителем**
03.02.2025	Инструктаж по технике безопасности. Ознакомление с нормативной документацией по технике безопасности, трудовому законодательству, внутренними документами организации.	
03.02.2025-19.02.2025	Изучение курса «Использование .NET API nanoCAD»: модули 1-2. Запуск первой команды, доработка функций запроса и вывода данных, посторенние отрезков, использование транзакций, работа со сложными и составными объектами.	
20.02.2025-05.03.2025	Изучение курса «Использование .NET API nanoCAD»: модули 3-4. Доработка функций построения различных геометрических элементов, работа с	

	разными системами координат, интерактивное создание и редактирование примитивов.	

6.03.2025-18.03.2025	Изучение курса «Использование .NET API nanoCAD»: модули 5-6. Работа с реакторами и событиями, модальными и немодальными окнами, а также палитрами.	
19.03.2025-02.04.2025	Изучение курса «Использование .NET API nanoCAD»: модули 7-8. Работа с расширенными данными, таблицами, словарями, переопределение характеристик стандартных примитивов. Введение в MultiCad.	
03.04.2025-17.04.2025	Изучение JavaScript: основы DOM, обработка событий, асинхронность. Практика написания интерактивных элементов веб-страниц.	
18.04.2025-02.05.2025	Освоение фреймворка React: компоненты, состояние, хуки. Разработка первых приложений, маршрутизация. Работа в Figma: создание макетов, использование сеток, компонентов, шрифтов и цветов. Подготовка дизайна для адаптивной верстки. Проектирование макетов сайта: страницы главная, команда, проекты. Совместная работа с другим frontend-разработчиком. UI/UX-оптимизация.	

03.05.2025-13.05.2025	Адаптивная верстка на React: интеграция макетов, разработка компонентов, отображение данных.	
14.05.2025-26.05.2025	Доработка и тестирование проекта: проверка отображения на разных устройствах, исправление ошибок, улучшение пользовательского опыта.	

Руководитель практики от образовательной организации:
Сивцев А. О.

подпись дата

Руководитель практики от профильной организации:
Сивцев А. О.

подпись дата

СОДЕРЖАНИЕ

1 РАЗРАБОТКА ПОГРАММНОГО МОДУЛЯ

1.1 Подготовка к работе

Для начала работы с модулем необходимо собранную библиотеку запустить в программе NanoCAD через Лента > Сервис > Приложения > Загрузка .NET приложения…

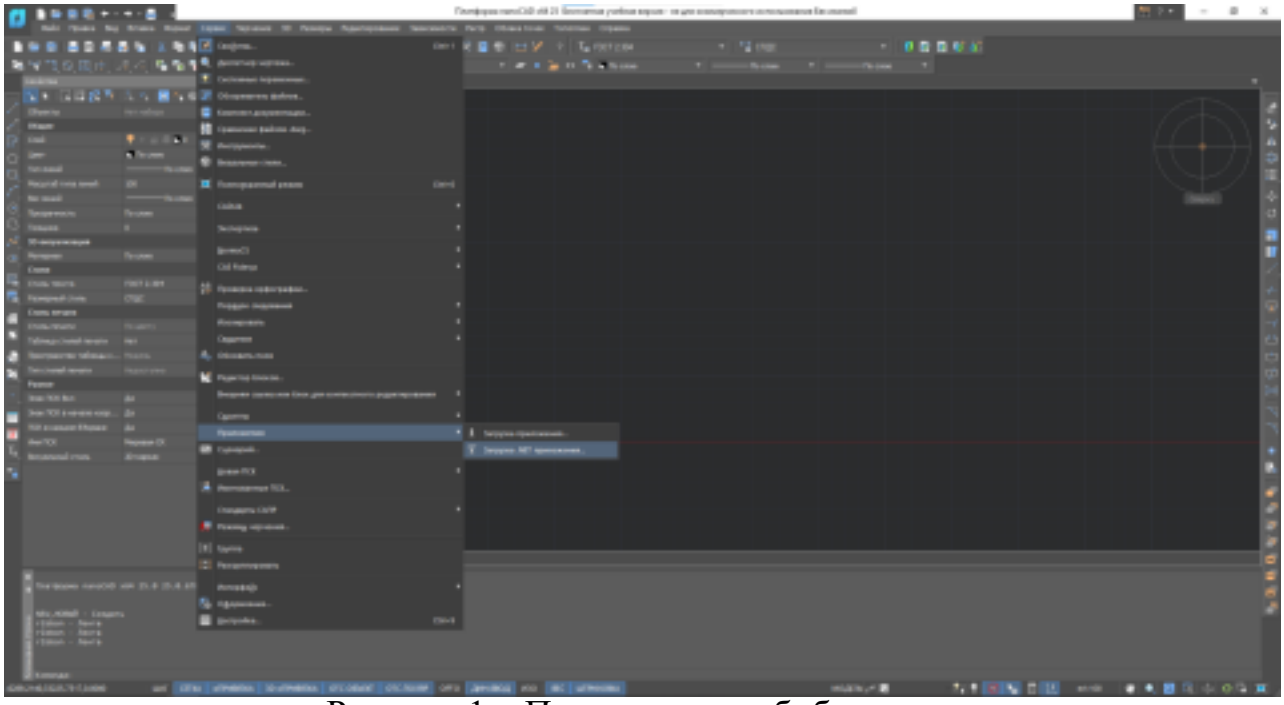

Рисунок 1 – Путь к запуску библиотеки

1.2 Модуль 1. Введение в .NET API.

В данном модуле было изучено: введение в .NET API; ввод данных; ключевые слова. Транзакции, создание объектов. Выбор, наборы выбора, приведение типов. Для начала была изучена возможность взаимодействия с командной строкой в программе NanoCAD путем вывода сообщения и принятия ввода, определение введенных данных, а именно их тип и использование ключевых слов. Изучена возможность создания отрезков и использования транзакций, примитивов (Рисунок 2, 3).

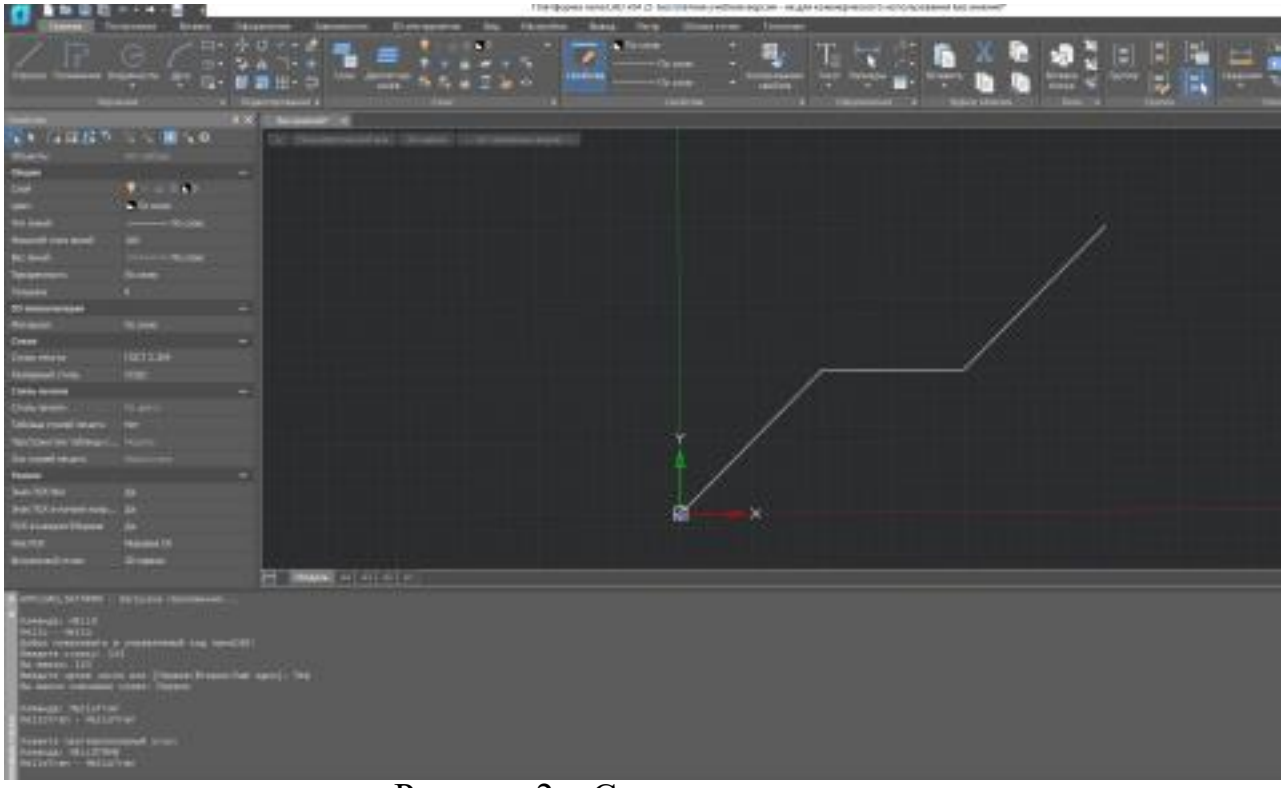

Рисунок 2 – Создание отрезков

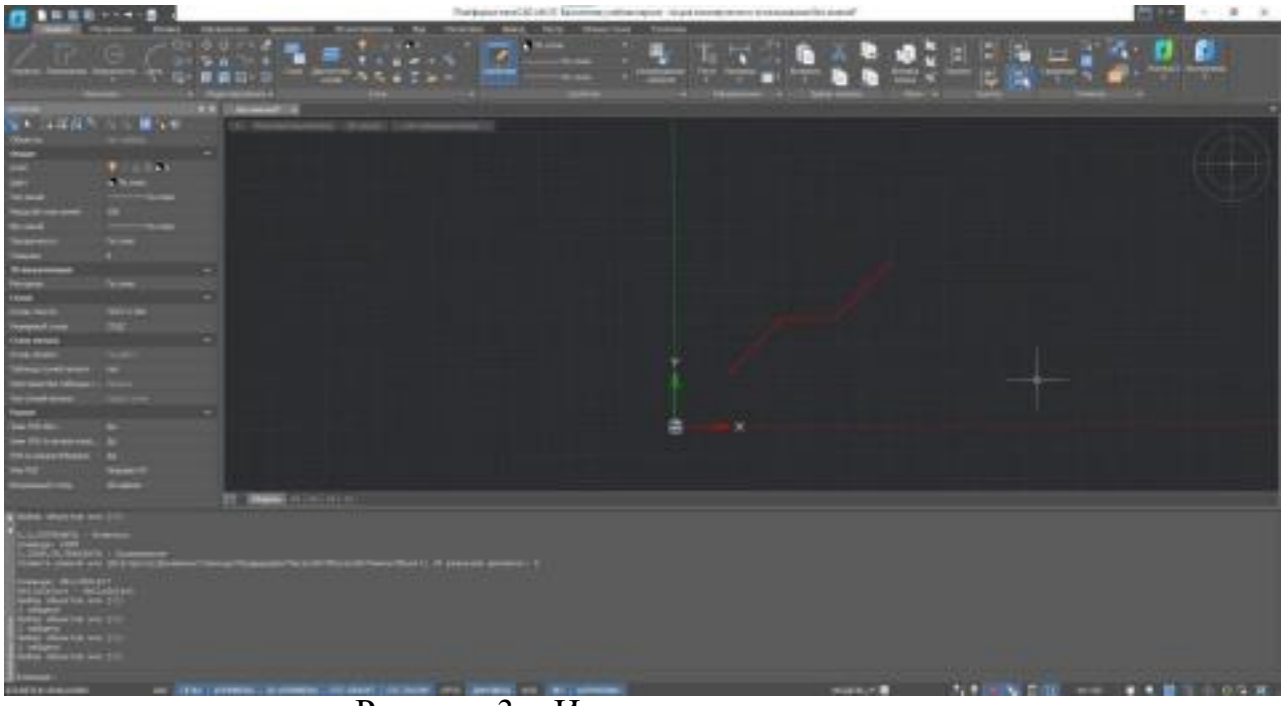

Рисунок 3 – Изменение примитивов

1.3 Модуль 2. Сложные составные объекты.

В данном модуле были изучены: сложные составные объекты; блоки,

полилинии и размеры. Сначала была создана команда, выводящая таблицу блоков, а также команда, добавляющая имеющийся объект в эту таблицу (Рисунок 4). Затем команда для вывода таблицы атрибутов (Рисунок 5). Программа, определяющая координаты вершин полилинии созданной командой PLINE (Рисунок 6). Программа для создания полилинии из точек, введенных пользователем нажатием мыши (Рисунок 7). Программа, определяющая координаты вершин 3д полилинии созданной командой 3DPOLY (Рисунок 8). Создание прямоугольника с размерами по двум соседним сторонам и измененным стилем (Рисунок 9).

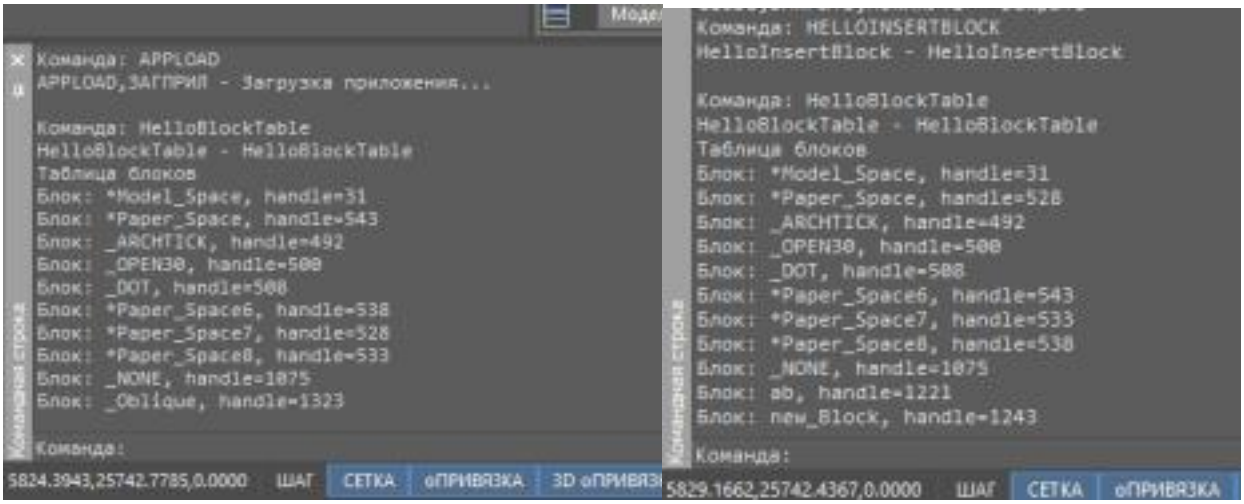

Рисунок 4 – Вывод таблицы блоков и добавление нового

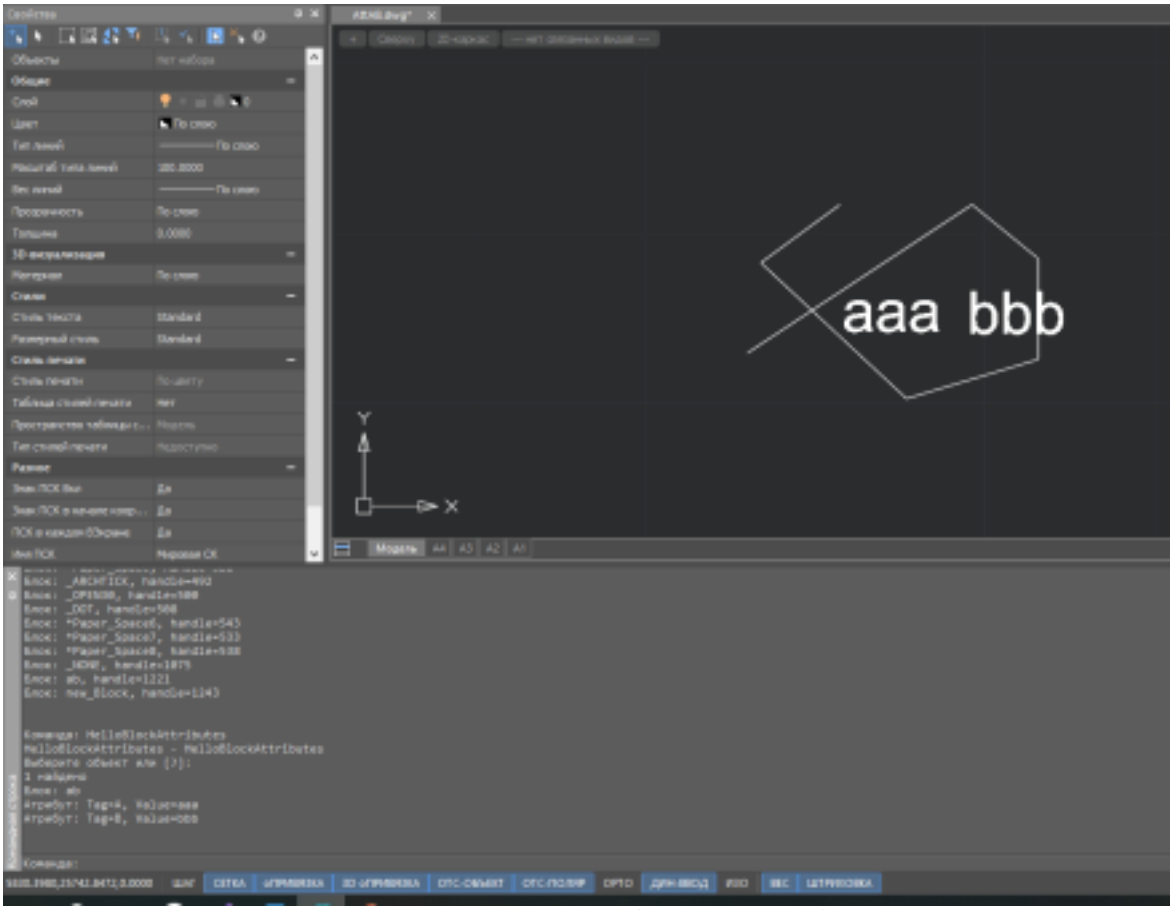

Рисунок 5 – Вывод таблицы атрибутов

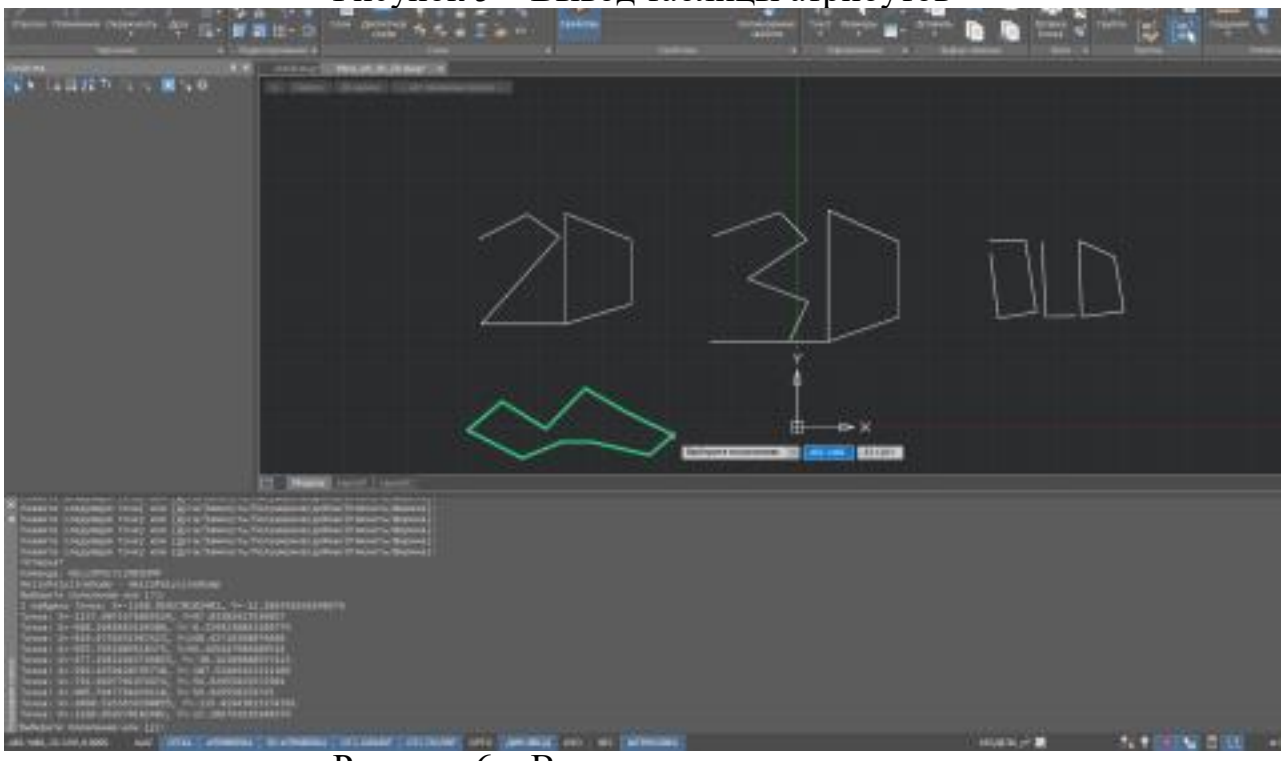

Рисунок 6 – Вывод вершин полилинии

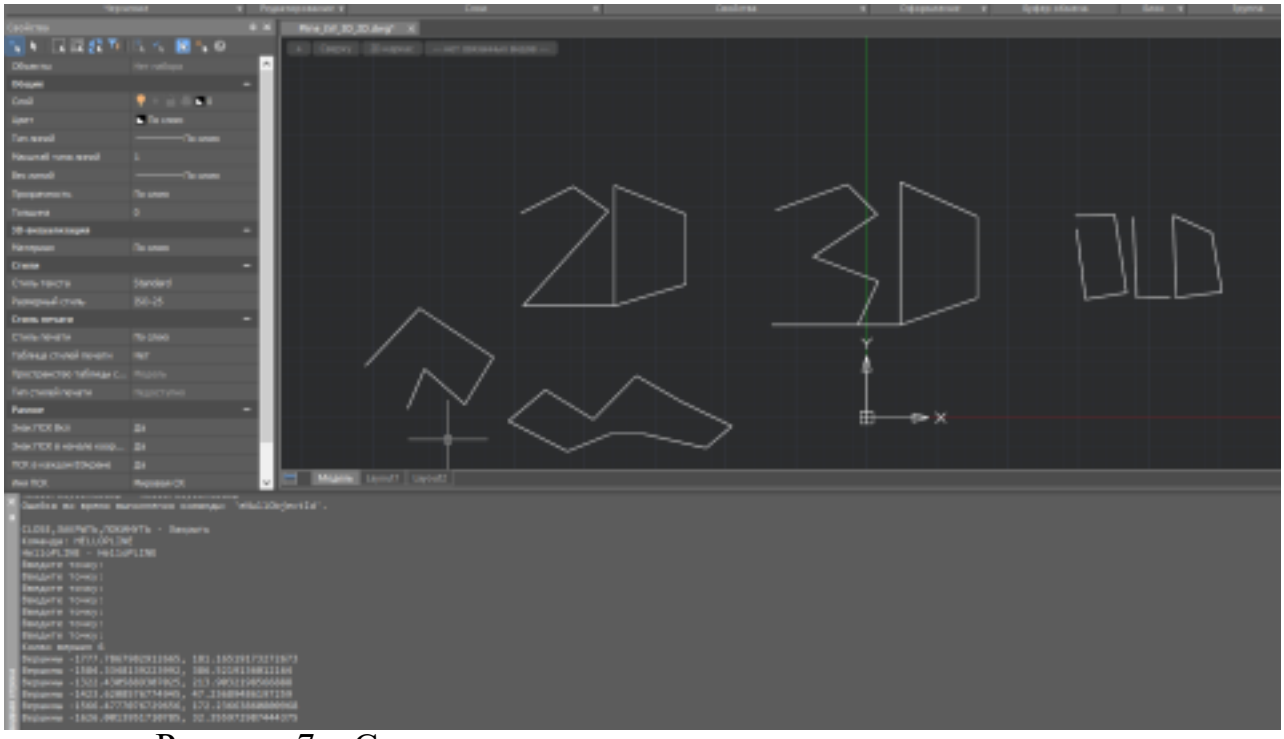

Рисунок 7 – Создание плилинии из пользовательскими точек

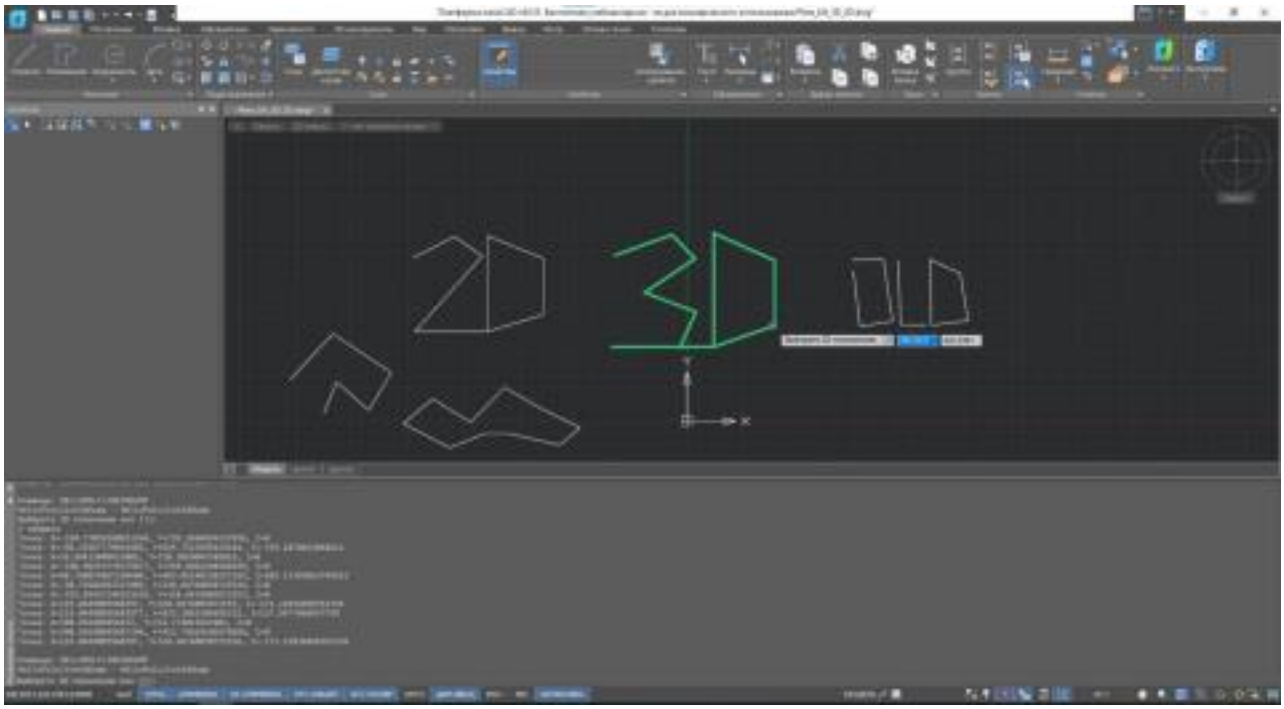

Рисунок 8 – Вывод вершин 3д полилинии

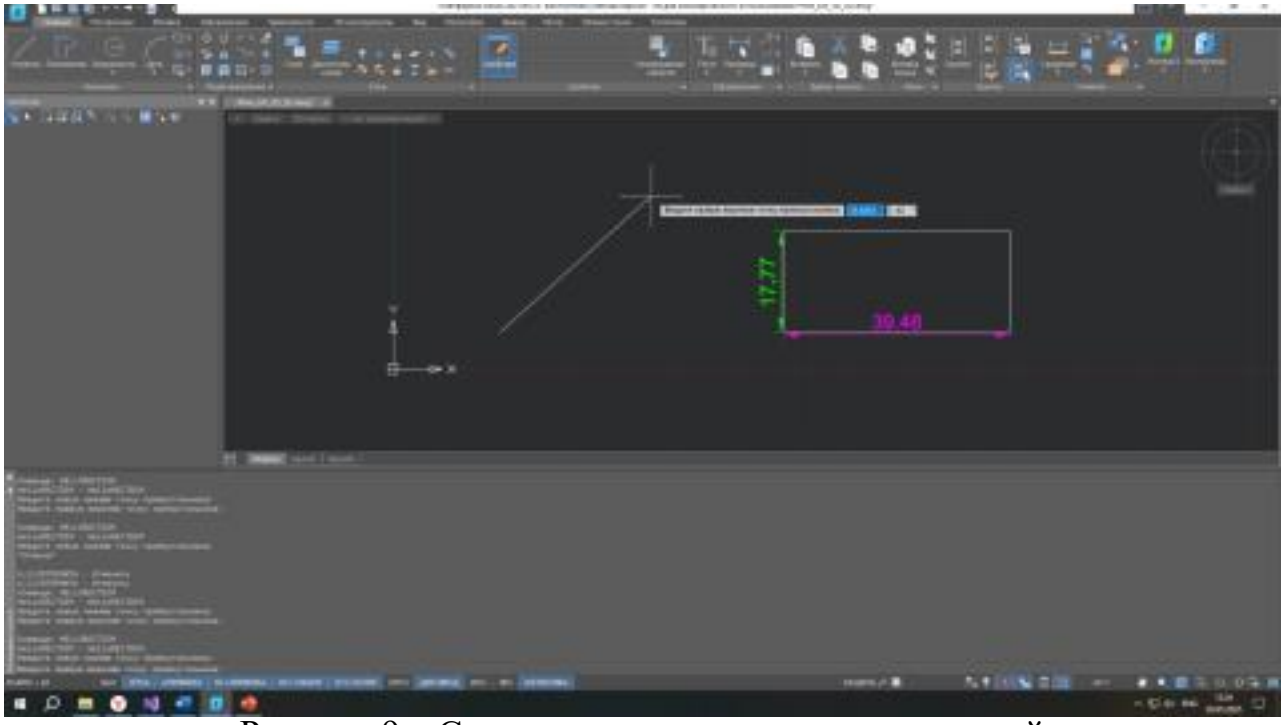

Рисунок 9 – Создание прямоугольника с метрикой

1.4 Модуль 3. Геометрическая библиотека, работа с кривыми. В данном модуле были изучены: геометрическая библиотека, работа с кривыми; координатные системы; команды в ПСК. Вывод длины вектора заданной пользователем двумя точками, вывод переводиться в дробные дюймы (Рисунок 10). Запрос у пользователя двух точек для построения линии и третей для проведения перпендикуляра к ней (Рисунок 11). Вывод количества и координат точек пересечения двух выбранных примитивов (Рисунок 12). Разбиение прямой на задаваемое количество отрезков (Рисунок 13). Создание прямоугольника по двум точкам в трехмерной системе координат (Рисунок 14). Вращение объекта вокруг оси на заданный градус (Рисунок 15). Выбор полилинии и точки в любом пространстве, определение лежит ли точка в одной плоскости с полилинией (Рисунок 16).

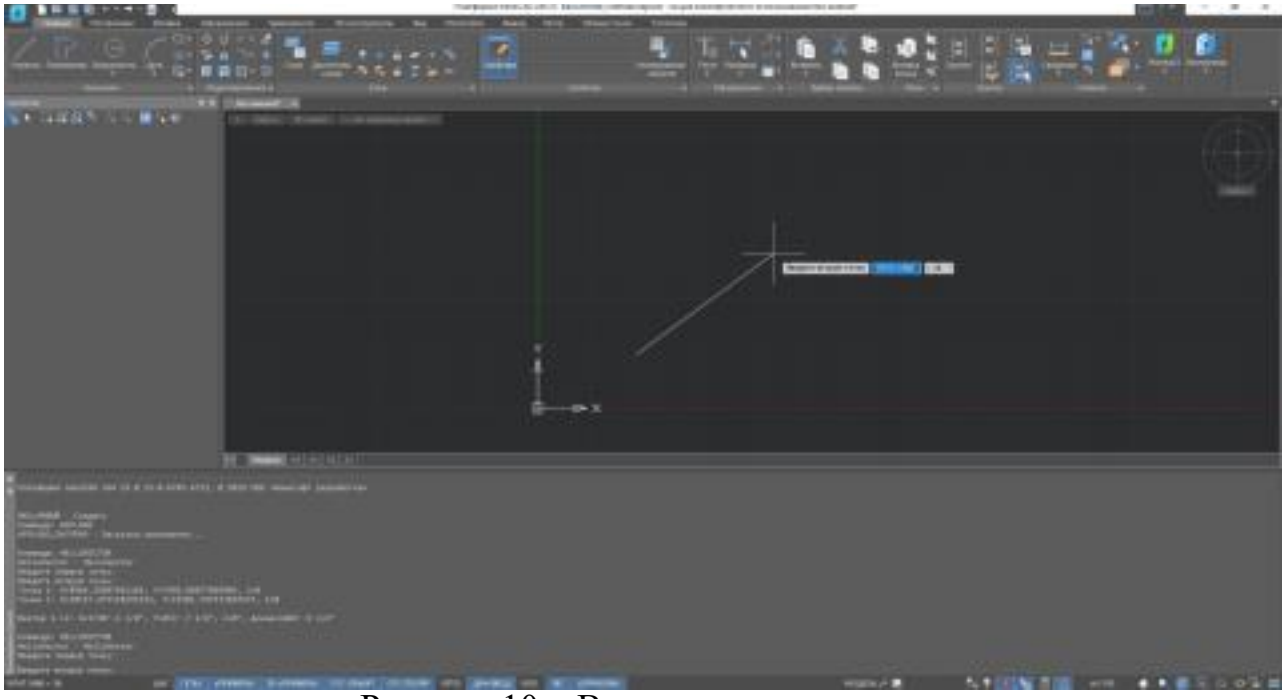

Рисунок 10 – Вывод длины вектора

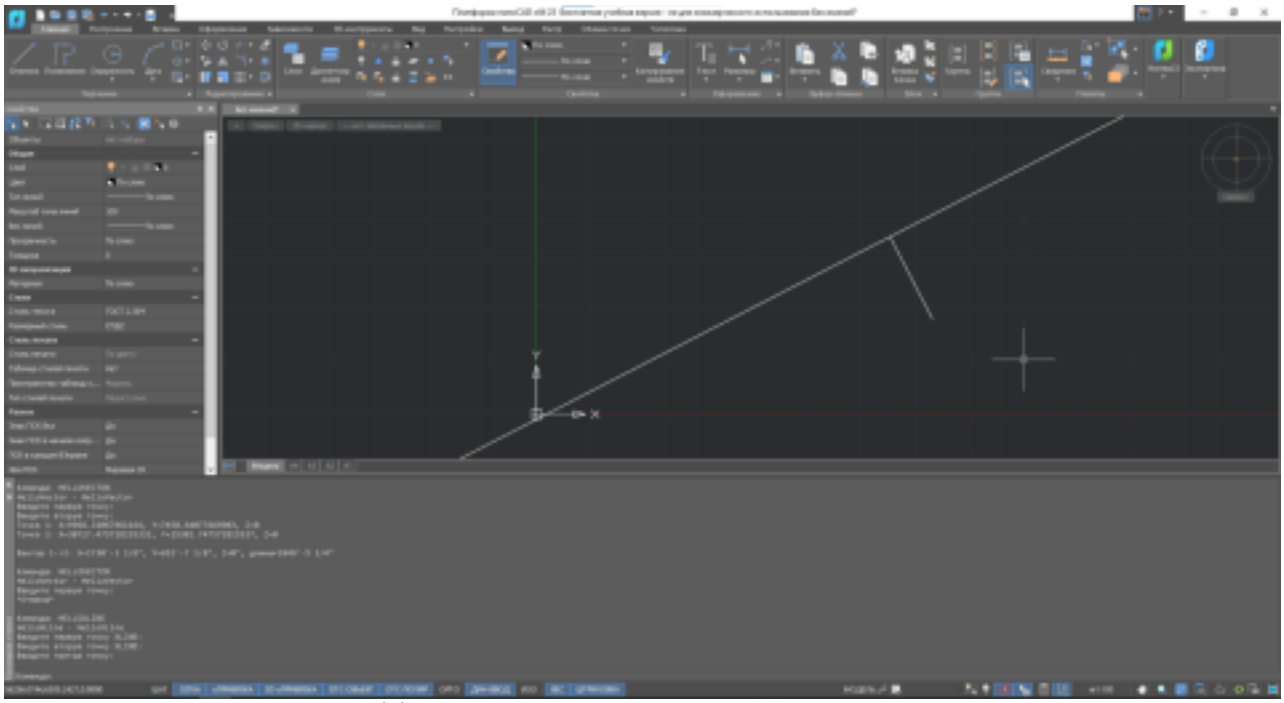

Рисунок 11 – Вывод линии и перпендикуляра к ней

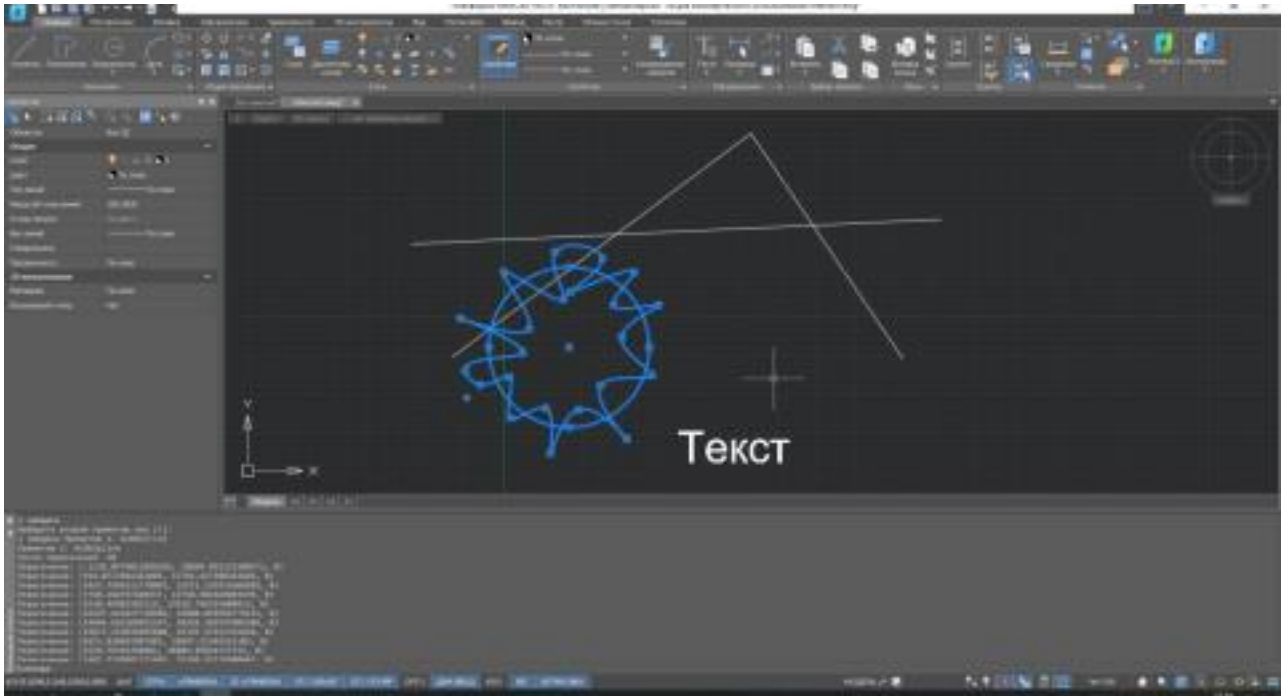

Рисунок 12 – Вывод кол-ва пересечений примитивов

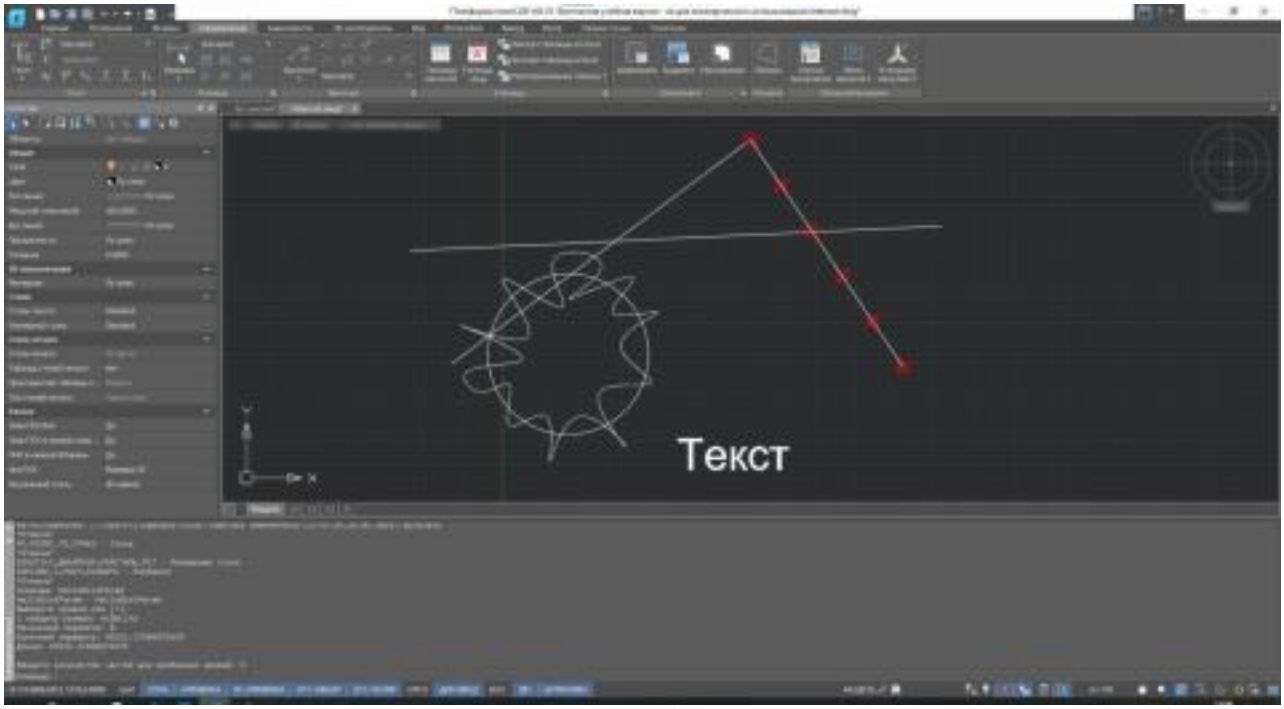

Рисунок 13 – Разбиение прямой

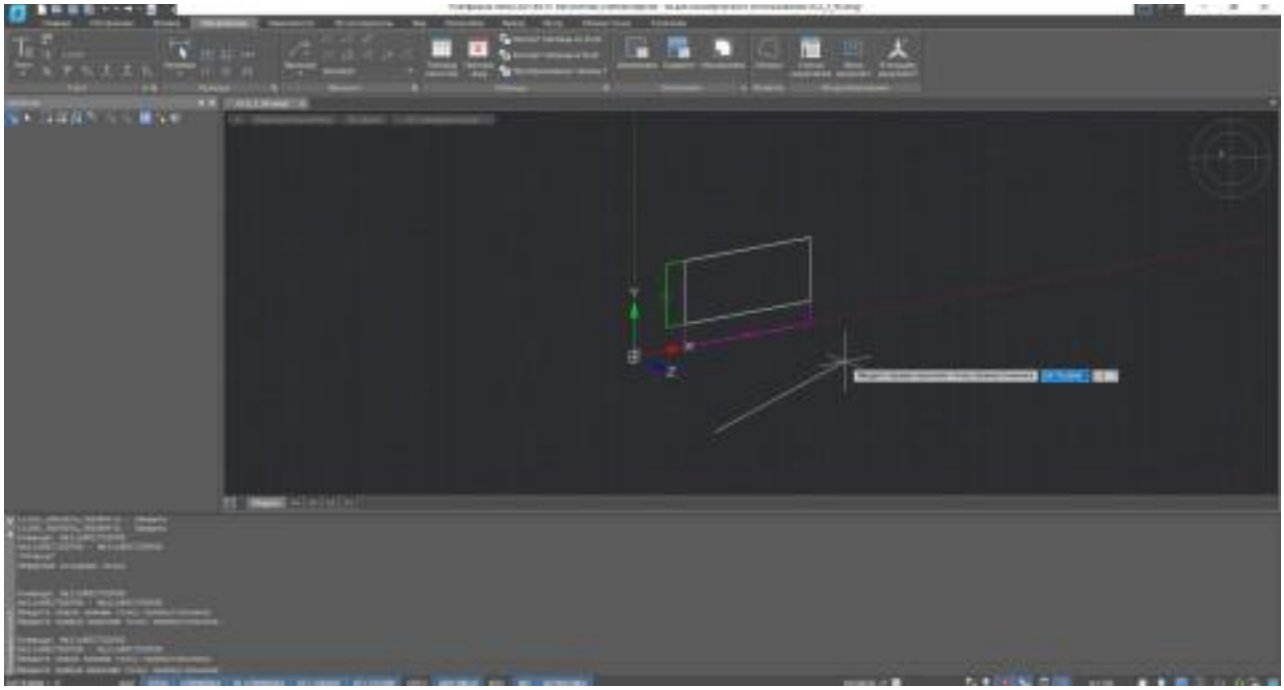

Рисунок 14 – Создание прямоугольника

Рисунок 15 – Вращение прямоугольника

Рисунок 16 – Определение плоскости точки

1.5 Модуль 4. Основные свойства примитивов.

В данном модуле были изучены: основные свойства примитивов; интерактивное создание и редактирование примитивов: Jig. Вывод названия и тип объектов содержимого словаря (Рисунок 17). Добавление нового стиля печати в словарь стилей печати, вывод список всех вхождений словаря стилей печати и изменение выбранного примитива. (Рисунок 18). Создание прямоугольника во двум точкам с предпросмотром (Рисунок 19). Создание прямоугольника с диагональю и размером двух сторон (Рисунок 20).

Рисунок 17 – Определение плоскости точки

Рисунок 18 – Создание нового стиля

Рисунок 19 – Создание прямоугольника с предпросмотром

Рисунок 20 – Создание прямоугольника с метрикой

1.6 Модуль 5. Реакторы.

В данном модуле были изучены: реакторы; события базы данных,

редактора, системы; контроль объектов, находящихся под курсором. Добавление реакторов и вывод при изменении координат заданного отрезка (Рисунок 21). Вывод нового и старого значения при изменении значения переменной DIMSCALE (Рисунок 22). Вывод сообщения при перемещении на лист чертежа (Рисунок 23). Программа рисует окружности по координатам курсора (Рисунок 24).

Рисунок 21 – Перемещение отрезка

Рисунок 22 – Изменение значений переменной

Рисунок 23 – Рисование окружностей

Рисунок 24 – Рисование окружностей

1.7 Модуль 6. Пользовательский интерфейс. Подсистема печати

В данном модуле были изучены: палитра(palette) WPF (Рисунок 30) и WinForms (Рисунок 29); модальные (диалоговое окно блокирует все остальные окна приложения до тех пор, пока оно не будет закрыто) и немодальные окна (диалоговое окно не блокирует другие окна приложения и допускает переключение на них) WPF и WinForms (Рисунок 25-28), управление листами и форматами через API.

Рисунок 25 – модальное окно WinForms

Рисунок 26 – модальное окно WPF

Рисунок 27 – модальное окно WinForms

Рисунок 28 – немодальное окно WPF

Рисунок 29 – палитра WinForms

Рисунок 30 – палитра WPF

Рисунок 31 – сохранение листа через команду HelloPlot

ЗАКЛЮЧЕНИЕ

В рамках текущей учебной практики были успешно выполнены следующие задачи:

1. Изучение курса «Программирование в NanoCAD, .NET API»: модули 1-2. Создание проекта, изучение ключевых слов, ввода данных и простейших преобразований объектов. Изучение блоков, полилиний, сетей и размеров точки.

2. Изучение курса «Программирование в NanoCAD, .NET API»: модули 3- 4. Изучение геометрической библиотеки и системы координат, работа с кривыми. Изучение стандартных свойств примитивов.

3. Изучение курса «Программирование в NanoCAD, .NET API»: модули 5. Изучение реакторов и событий. 6. Пользовательские интерфейсы. Подсистема печати.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. SDK NanoCAD

Документация по API ObjectARX от Autodesk:

2. URL: https://help.autodesk.com/view/OARX/2022/ENU/

Документация по API .NET от Autodesk:

3. URL: https://help.autodesk.com/view/OARX/2025/ENU/

4. Конференция разработчиков nanoCAD: всё об API Платформы nanoCAD из первых уст(общая сессия и секция1) // RuTube. – URL: https://rutube.ru/video/8ae54678efc2b25d2590f33621734ef7/ (дата обращения: 15.03.2025).

5. URL: https://habr.com/ru/companies/nanosoft/news/894682/

Документация по API .NET от Autodesk:

6. URL: https://help.autodesk.com/

Руководство пользования API от Autodesk

ПРИЛОЖЕНИЕ А. ПОЛНЫЙ ЛИСТИНГ ПРОГРАММНОГО МОДУЛЯ Код файла Class1.cs

```
namespace nanoApplicationNET {

 using Teigha.Runtime;
 using HostMgd.EditorInput;
 using Teigha.DatabaseServices;
 using Teigha.Geometry;
 using HostMgd.ApplicationServices;

 using Platform = HostMgd;
 using PlatformDb = Teigha;
 using System.Windows.Media.Media3D.Converters;
 using HostMgd.Runtime;
 using System.Windows.Media.Media3D;

 using Teigha.Colors;
 using System.Xml.Linq;


 public class Class1
 {
 //[CommandMethod("Hello")]
 //public void Hello()
 //{
 // Editor ed =
HostMgd.ApplicationServices.Application.DocumentManager.MdiActiveDocument.Editor;  //

ed.WriteMessage("Добро пожаловать в управляемый код nanoCAD!");

 // PromptStringOptions opts = new PromptStringOptions("Введите строку:");
 // opts.AllowSpaces = true;
 // PromptResult pr = ed.GetString(opts);

 // if (PromptStatus.OK == pr.Status)
 // {
 // ed.WriteMessage("Вы ввели: " + pr.StringResult);  // }
 // else
 // {
 // ed.WriteMessage("Отмена.");
 // }


 // PromptIntegerOptions iopts = new PromptIntegerOptions("Введите целое
число:");
 // iopts.Keywords.Add("Первое");
 // iopts.Keywords.Add("Второе");
 // iopts.Keywords.Add("Ещё одно");
 // iopts.AppendKeywordsToMessage = true;

 // PromptIntegerResult ipr = ed.GetInteger(iopts);

 // if (PromptStatus.OK == ipr.Status)
 // {
 // ed.WriteMessage("Вы ввели: " + ipr.Value);
```



```
 // }
 // if (PromptStatus.Keyword == ipr.Status)
 // {
 // ed.WriteMessage("Вы ввели ключевое слово: " +
```

```
ipr.StringResult);
 // }
 // else
 // {
 // ed.WriteMessage("Отмена");
 // }

 //}

 //[CommandMethod("HelloTran")]
 //public void HelloTran()
 //{
 // Document doc = Application.DocumentManager.MdiActiveDocument; //  Получение
ссылки на активный документ
 // Editor ed = doc.Editor; // Получение ссылки на редактор докумена  // Database
db = doc.Database; // Получение ссылки на базу данных  документа
 // ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать, если  нужно
добавлять объекты в текущий лист, вместо ModelSpace

 // using (Transaction trans =
db.TransactionManager.StartTransaction())
 // {
 // BlockTable blockTable = trans.GetObject(db.BlockTableId,  OpenMode.ForRead)
as BlockTable; // Получение ссылки на таблицу блоков в базе данных документа
 // BlockTableRecord modelSpace =
trans.GetObject(blockTable[BlockTableRecord.ModelSpace], OpenMode.ForWrite) as
BlockTableRecord; // Получение ссылки на пространство модели документа

 // // TODO 1.2.1: Создать отрезок (Line)
 // Line line = new Line(new Point3d(0, 0, 0), new Point3d(100,  100, 0));
 // Line line2 = new Line(new Point3d(100, 100, 0), new  Point3d(200,
100, 0));
 // Line line3 = new Line(new Point3d(200, 100, 0), new  Point3d(300,
200, 0));
 // // TODO 1.2.2: Добавить в блок modelSpace
 // modelSpace.AppendEntity(line);
 // modelSpace.AppendEntity(line2);
 // modelSpace.AppendEntity(line3);
 // // TODO 1.2.3: Добавить в транзакцию
 // trans.AddNewlyCreatedDBObject(line, true);
 // trans.AddNewlyCreatedDBObject(line2, true);  //
trans.AddNewlyCreatedDBObject(line3, true);

 // trans.Commit(); // или trans.Abort();
 // }
 //}


 //[CommandMethod("HelloSelect")]
 //public void HelloSelect()
 //{
 // Document doc = Application.DocumentManager.MdiActiveDocument; //  Получение
ссылки на активный документ
 // Editor ed = doc.Editor; // Получение ссылки на редактор докумена
```



```
 // Database db = doc.Database; // Получение ссылки на базу данных  документа
 // ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать, если  нужно
добавлять объекты в текущий лист, вместо ModelSpace

 // PromptSelectionOptions opts; // Параметры для диалога с
пользователем
 // PromptSelectionResult pr; // Переменная для результата диалога с  пользователем

 // opts = new PromptSelectionOptions();
 // opts.MessageForAdding = "Выберите объекты: ";
```

```
 // pr = ed.GetSelection();

 // SelectionSet ss = pr.Value;

 // using (Transaction trans =
db.TransactionManager.StartTransaction())
 // {
 // foreach (SelectedObject ssobj in ss) // Перебор объектов,  выбранных
пользователем
 // {
 // ObjectId objectId = ssobj.ObjectId; // Присвоение  переменной objectId
значения ID выбранного объекта
 // DBObject dbobj = trans.GetObject(objectId,  OpenMode.ForWrite); //
Открытие объекта для изменения из набора выбранных  пользователем объектов
 // Entity ent = dbobj as Entity; // Приведение объекта к типу  Entity

 // if (ent != null)
 // {
 // // TODO 1.3.1: Задать красный цвет (свойство  ColorIndex)
 // ent.ColorIndex = 1; // Красный


 // Matrix3d mat = Matrix3d.Displacement(new Vector3d(100,  100, 0));


 // //Matrix3d mat = Matrix3d.Rotation(Math.PI/4,  Vector3d.ZAxis,
Point3d.Origin);
 // //Matrix3d mat = Matrix3d.Scaling(2, Point3d.Origin);  // //Matrix3d mat =
Matrix3d.Mirroring(new  Line3d(Point3d.Origin, Vector3d.YAxis));

 // // TODO 1.3.2: Преобразовать примитив при помощи  матрицы TransformBy(mat)
 // ent.TransformBy(mat);
 // }
 // }

 // trans.Commit();
 // }
 //}


 //2 модуль Blocks
```



```
 //[CommandMethod("HelloBlockTable")]
 //public void HelloBlockTable()
 //{
 // Document doc = Application.DocumentManager.MdiActiveDocument;  // Editor
ed = doc.Editor;
 // Database db = doc.Database;
 // ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать, если нужно
добавлять объекты в текущий лист, вместо ModelSpace

 // using (Transaction trans =
db.TransactionManager.StartTransaction())
 // {
 // BlockTable blockTable = trans.GetObject(db.BlockTableId,  OpenMode.ForRead)
as BlockTable; // Получение ссылки на таблицу блоков в базе данных документа

 // ed.WriteMessage("\nТаблица блоков\n");

 // foreach (ObjectId blockId in blockTable) // Перебор ID блоков из таблицы блоков
blockTable
 // {
 // DBObject obj = trans.GetObject(blockId, OpenMode.ForRead,  false, true); //
```

Открытие очередного блока на чтение

```
 // BlockTableRecord btr = obj as BlockTableRecord; //  Приведение открытого
объекта к типу BlockTableRecord

 // if (btr != null)
 // {
 // string blockName = btr.Name; // TODO 2.1.1  // long handle =
btr.Handle.Value; // TODO 2.1.1

 // ed.WriteMessage("Блок: {0}, handle={1}\n", blockName,  handle);
 // }
 // }

 // trans.Commit();
 // }
 //}


 //[CommandMethod("HelloInsertBlock")]
 //public void HelloInsertBlock()
 //{
 // Document doc = Application.DocumentManager.MdiActiveDocument;  // Editor
ed = doc.Editor;
 // Database db = doc.Database;
 // ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать, если нужно
добавлять объекты в текущий лист, вместо ModelSpace

 // string srcFileName = "D:\\visual studio
projekts\\repos\\PD\\doto\\Module2\\AttAB.dwg"; // TODO 2.1.2: Запросить у
пользователя полный путь к файлу-источнику
 // string dstBlockName = "new_Block"; // TODO 2.1.2: Задать название нового блока

 // using (Database srcDb = new Database(false, false))  // {
 // srcDb.ReadDwgFile(srcFileName,
FileOpenMode.OpenForReadAndAllShare, true, "");
```



```
// ObjectId blockId = db.Insert("*Model_Space", dstBlockName,  srcDb, false);
 // }
 //}




 //[CommandMethod("HelloBlockIterator")]
 //public void HelloBlockIterator()
 //{
 // Document doc = Application.DocumentManager.MdiActiveDocument;  // Editor
ed = doc.Editor;
 // Database db = doc.Database;
 // ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать, если нужно
добавлять объекты в текущий лист, вместо ModelSpace

 // using (Transaction trans =
db.TransactionManager.StartTransaction())
 // {
 // BlockTable blockTable = trans.GetObject(db.BlockTableId,
OpenMode.ForRead) as BlockTable;

 // PromptStringOptions opts = new PromptStringOptions("Введите имя блока(new_Block):
");
 // opts.AllowSpaces = true;
 // PromptResult UserBlockName = ed.GetString(opts);  // // TODO 2.1.3:
Запросить имя блока для перебора вместо  ModelSpace
```

```
 // // TODO 2.1.3: Заменить blockTable[BlockTableRecord.ModelSpace]  на
blockTable[UserBlockName]
 // BlockTableRecord btr =
trans.GetObject(blockTable[UserBlockName.StringResult], OpenMode.ForRead) as
BlockTableRecord; // Получение ссылки на таблицу блоков в базе данных документа

 // ed.WriteMessage("\nБлок: {0}\n", btr.Name);

 // foreach (ObjectId objectId in btr)
 // {
 // DBObject obj;

 // obj = trans.GetObject(objectId, OpenMode.ForRead, false,  true);

 // string className = obj.GetRXClass().Name;  // long handle =
obj.Handle.Value;

 // ed.WriteMessage("Объект: {0}, handle={1}\n", className,  handle);
 // }

 // trans.Commit();
 // }
 //}



 //[CommandMethod("HelloBlockAttributes")]
 //public void HelloBlockAttributes()
```



```
 //{
 // Document doc = Application.DocumentManager.MdiActiveDocument;  // Editor
ed = doc.Editor;
 // Database db = doc.Database;
 // ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать, если нужно
добавлять объекты в текущий лист, вместо ModelSpace

 // // TODO 2.1.4: Попросить пользователя указать блок на чертеже  // ObjectId
blockId = ed.GetEntity("Выберите объект: ").ObjectId;

 // using (Transaction trans =
db.TransactionManager.StartTransaction())
 // {
 // BlockReference block = trans.GetObject(blockId,
OpenMode.ForRead) as BlockReference;

 // ed.WriteMessage("\nБлок: {0}\n", block.Name);

 // if (block != null)
 // {
 // AttributeCollection attributes = block.AttributeCollection;

 // foreach (ObjectId attrId in attributes)  // {
 // AttributeReference? attrRef = trans.GetObject(attrId,  OpenMode.ForRead) as
AttributeReference;

 // if (attrRef != null)
 // {
 // string tag = attrRef.Tag;
 // string value = attrRef.TextString;  // ed.WriteMessage($"Атрибут:
Tag={tag},  Value={value}\n");
 // }
 // }
 // }
 // else
```

```
 // {
 // ed.WriteMessage("Не удалось открыть блок\n");  // }

 // trans.Commit();
 // }
 //}


 ////2 модуль Polylines


 //[CommandMethod("HelloPolylineDump")]
 //public void HelloPolylineDump()
 //{
 // Document doc = Application.DocumentManager.MdiActiveDocument;  // Editor
ed = doc.Editor;
 // Database db = doc.Database;
 // ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать, если нужно
добавлять объекты в текущий лист, вместо ModelSpace
 // ObjectId plineId;

 // PromptEntityOptions opts;
 // PromptEntityResult pr;
```



```
 // opts = new PromptEntityOptions("Выберите полилинию: ");  // pr =
ed.GetEntity(opts);

 // plineId = pr.ObjectId;

 // using (Transaction trans =
db.TransactionManager.StartTransaction())
 // {
 // Polyline pline = trans.GetObject(plineId, OpenMode.ForRead) as  Polyline; // TODO
2.2.1: Открыть полилинию на чтение и привести к типу Polyline

 // if (pline != null)
 // {
 // for (int i = 0; i < pline.NumberOfVertices; i++)  // {
 // Point2d p = pline.GetPoint2dAt(i);
 // ed.WriteMessage("Точка: X={0}, Y={1}\n", p.X, p.Y);  // }
 // }
 // else
 // {
 // ed.WriteMessage("Не удалось открыть полилинию\n");  // }

 // trans.Commit();
 // }
 //}




 //[CommandMethod("HelloPLINE")]
 //public void HelloPLINE()
 //{
 // Document doc = Application.DocumentManager.MdiActiveDocument;  // Editor
ed = doc.Editor;
 // Database db = doc.Database;
 // ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать, если нужно
добавлять объекты в текущий лист, вместо ModelSpace

 // Point3dCollection points = new Point3dCollection();  // Polyline
pline = null;
```

```
 // while (true)
 // {
 // PromptPointOptions opts = new PromptPointOptions("Введите точку: ");
 // opts.AllowNone = true;

 // PromptPointResult pr = ed.GetPoint(opts);

 // if (PromptStatus.OK == pr.Status)
 // {
 // points.Add(pr.Value);
 // }
 // else
 // {
 // break;
 // }
```



```
// }

 // if (points.Count > 0)
 // {
 // pline = new Polyline(); // TODO 2.2.2: Создать объект типа Polyline
 // ed.WriteMessage("Колво вершин {0}", points.Count);

 // for (int i = 0; i < points.Count; i++)
 // {
 // pline.AddVertexAt(0, new Point2d(points[i].X, points[i].Y),  0.0, 0.0, 0.0);
 // ed.WriteMessage("Вершины {0}, {1}", points[i].X,  points[i].Y);
 // }
 // // TODO 2.2.2: Добавить в цикле for точки в полилинию при  помощи
pline.AddVertexAt()
 // }

 // if (pline != null)
 // {
 // using (Transaction trans =
db.TransactionManager.StartTransaction())
 // {
 // BlockTable blockTable = trans.GetObject(db.BlockTableId,  OpenMode.ForRead) as
BlockTable;
 // BlockTableRecord modelSpace =
trans.GetObject(blockTable[BlockTableRecord.ModelSpace], OpenMode.ForWrite) as
BlockTableRecord;

 // modelSpace.AppendEntity(pline);
 // trans.AddNewlyCreatedDBObject(pline, true);

 // trans.Commit();
 // }
 // }
 //}




 //[CommandMethod("HelloPolyline3dDump")]
 //public void HelloPolyline3dDump()
 //{
 // Document doc = Application.DocumentManager.MdiActiveDocument;  // Editor
ed = doc.Editor;
 // Database db = doc.Database;
 // ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать, если нужно
добавлять объекты в текущий лист, вместо ModelSpace
 // ObjectId plineId;
```

```
 // PromptEntityOptions opts;
 // PromptEntityResult pr;

 // opts = new PromptEntityOptions("Выберите 3D полилинию: ");  // pr =
ed.GetEntity(opts);

 // plineId = pr.ObjectId;

 // using (Transaction trans =
db.TransactionManager.StartTransaction())
```



```
 // {
 // Polyline3d pline = trans.GetObject(plineId, OpenMode.ForRead)  as Polyline3d;

 // if (pline != null)
 // {
 // foreach (ObjectId vertexId in pline)
 // {
 // PolylineVertex3d? vertex = trans.GetObject(vertexId,  OpenMode.ForRead) as
PolylineVertex3d;
 // Point3d position = vertex.Position;  // ed.WriteMessage("\nТочка: X={0},
Y={1}, Z={2}",  position.X, position.Y, position.Z);
 // }
 // // TODO 2.2.3: Вывести в командную строку список вершин 3D  полилинии
 // // Перебрать вершины в цикле foreach
 // // Вывод координат - PolylineVertex3d.Position;

 // }
 // else
 // {
 // ed.WriteMessage("Не удалось открыть 3D полилинию\n");  // }

 // trans.Commit();
 // }
 //}




 //[CommandMethod("Hello3DPOLY")]
 //public void Hello3DPOLY()
 //{
 // Document doc = Application.DocumentManager.MdiActiveDocument;  // Editor
ed = doc.Editor;
 // Database db = doc.Database;
 // ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать, если нужно
добавлять объекты в текущий лист, вместо ModelSpace

 // Point3dCollection points = new Point3dCollection();  //
Polyline3d pline = null;

 // while (true)
 // {
 // PromptPointOptions opts = new PromptPointOptions("Введите точку: ");
 // opts.AllowNone = true;

 // PromptPointResult pr = ed.GetPoint(opts);

 // if (PromptStatus.OK == pr.Status)
 // {
 // points.Add(pr.Value);
 // }
 // else
 // {
```

```
 // break;
 // }
 // }
```



```
 // if (points.Count > 0)
 // {
 // pline = new Polyline3d(Poly3dType.SimplePoly, points, true);  // // TODO
2.2.4: Создать объект типа Polyline3d, используя Poly3dType.SimplePoly

 // }

 // if (pline != null)
 // {
 // using (Transaction trans =
db.TransactionManager.StartTransaction())
 // {
 // BlockTable blockTable = trans.GetObject(db.BlockTableId,  OpenMode.ForRead) as
BlockTable;
 // BlockTableRecord modelSpace =
trans.GetObject(blockTable[BlockTableRecord.ModelSpace], OpenMode.ForWrite) as
BlockTableRecord;

 // modelSpace.AppendEntity(pline);
 // trans.AddNewlyCreatedDBObject(pline, true);

 // trans.Commit();
 // }
 // }
 //}


 ////2 модуль Dimensions


 //[CommandMethod("HelloDimStyle")]
 //public void HelloDimStyle()
 //{
 // Document doc = Application.DocumentManager.MdiActiveDocument;  // Editor
ed = doc.Editor;
 // Database db = doc.Database;
 // ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать, если нужно
добавлять объекты в текущий лист, вместо ModelSpace

 // ObjectId dimTableId = db.DimStyleTableId;
 // ObjectId dimStyleId;

 // using (Transaction trans =
db.TransactionManager.StartTransaction())
 // {
 // DimStyleTable dimStyleTable = trans.GetObject(dimTableId,
OpenMode.ForWrite) as DimStyleTable;
 // dimStyleId = dimStyleTable["Standard"];

 // DimStyleTableRecord dimStyleTableRecord =
trans.GetObject(dimStyleId, OpenMode.ForRead) as DimStyleTableRecord;

 // DimStyleTableRecord newDimStyleTableRecord = new
DimStyleTableRecord();
 // newDimStyleTableRecord.CopyFrom(dimStyleTableRecord);  //
newDimStyleTableRecord.Name = "NewStandard";

 // newDimStyleTableRecord.Dimscale = 50;
 // newDimStyleTableRecord.Dimdec = 1;
```

```
 // // TODO 2.3.1: Задать новые значения свойств Dimscale, Dimdec

 // ObjectId newDimStyleId =
dimStyleTable.Add(newDimStyleTableRecord);
 // trans.AddNewlyCreatedDBObject(newDimStyleTableRecord, true);  // db.Dimstyle =

newDimStyleId;

 // trans.Commit();
 // }
 //}




 //[CommandMethod("HelloRECTDIM")]
 //public void HelloRECTDIM()
 //{
 // Document doc = Application.DocumentManager.MdiActiveDocument;  // Editor
ed = doc.Editor;
 // Database db = doc.Database;
 // ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать, если нужно
добавлять объекты в текущий лист, вместо ModelSpace

 // Point3d p1, p2;
 // Point3dCollection points = new Point3dCollection();

 // PromptPointOptions opts = new PromptPointOptions("Введите левую нижнюю точку
прямоугольника: ");
 // PromptPointResult pr = ed.GetPoint(opts);

 // if (PromptStatus.OK == pr.Status)
 // {
 // p1 = pr.Value;

 // opts.Message = "Введите правую верхнюю точку прямоугольника: ";  //
opts.BasePoint = p1;
 // opts.UseBasePoint = true;

 // pr = ed.GetPoint(opts);
 // if (PromptStatus.OK == pr.Status)
 // {
 // p2 = pr.Value;

 // // Создать прямоугольник по точкам
 // points.Add(p1);
 // points.Add(new Point3d(p1.X, p2.Y, 0)); // Правая нижняя  точка
 // points.Add(p2);
 // points.Add(new Point3d(p2.X, p1.Y, 0)); // Левая верхняя  точка

 // Polyline pline = new Polyline(points.Count);

 // for (int i = 0; i < points.Count; i++)
 // {
 // pline.AddVertexAt(i, new Point2d(points[i].X,  points[i].Y), 0, 0, 0);
 // }

 // pline.Closed = true;
```



```
 // // Создать вертикальный размер
 // RotatedDimension dimver = new RotatedDimension();  // dimver.XLine1Point = p1;
// Левая нижняя точка  // dimver.XLine2Point = new Point3d(p1.X, p2.Y, 0); // Правая
нижняя точка
```

```
 // dimver.DimLinePoint = new Point3d(p1.X - 0.5, (p1.Y + p2.Y)  / 2, 0); // Середина
левой стороны
 // dimver.Rotation = Math.PI / 2; // Поворот на 90 градусов  // dimver.ColorIndex
= 3; // Зеленый цвет

 // // Создать горизонтальный размер
 // RotatedDimension dimhor = new RotatedDimension();  // dimhor.XLine1Point = p1;
// Левая нижняя точка  // dimhor.XLine2Point = new Point3d(p2.X, p1.Y, 0); // Левая
верхняя точка
 // dimhor.DimLinePoint = new Point3d((p1.X + p2.X) / 2, p1.Y - 0.5, 0); // Середина
нижней стороны
 // dimhor.ColorIndex = 6; // Розовый цвет

 // using (Transaction trans =
db.TransactionManager.StartTransaction())
 // {
 // BlockTable blockTable =
trans.GetObject(db.BlockTableId, OpenMode.ForRead) as BlockTable;
// BlockTableRecord modelSpace =
trans.GetObject(blockTable[BlockTableRecord.ModelSpace], OpenMode.ForWrite) as
BlockTableRecord;

 // modelSpace.AppendEntity(pline);
 // trans.AddNewlyCreatedDBObject(pline, true);

 // modelSpace.AppendEntity(dimver);
 // trans.AddNewlyCreatedDBObject(dimver, true);

 // modelSpace.AppendEntity(dimhor);
 // trans.AddNewlyCreatedDBObject(dimhor, true);

 // trans.Commit();
 // }
 // }
 // }
 //}



 //// 3 модуль Geometry

 //[CommandMethod("HelloVector")]
 //public void HelloVector()
 //{
 // Document doc = Application.DocumentManager.MdiActiveDocument;  // Editor
ed = doc.Editor;
 // Database db = doc.Database;
 // ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать, если нужно
добавлять объекты в текущий лист, вместо ModelSpace

 // Point3d p1, p2;

 // PromptPointOptions opts = new PromptPointOptions("Введите первую точку");
```



```
// PromptPointResult pr = ed.GetPoint(opts);

 // if (PromptStatus.OK != pr.Status)
 // return;

 // p1 = pr.Value;

 // opts.Message = "Введите вторую точку";
 // opts.BasePoint = p1;
 // opts.UseBasePoint = true;
```

```
 // pr = ed.GetPoint(opts);
 // if (PromptStatus.OK != pr.Status)
 // return;

 // p2 = pr.Value;

 // ed.WriteMessage("\nТочка 1: X={0}, Y={1}, Z={2}\n", p1.X, p1.Y,  p1.Z); //
TODO 3.1.3
 // ed.WriteMessage("Точка 2: X={0}, Y={1}, Z={2}\n", p2.X, p2.Y,  p2.Z); //
TODO 3.1.3


 // // TODO 3.1.1: Вычислить вектор перемещения из точки p1 в точку p2  и его длину
 // Vector3d v = p1.GetVectorTo(p2); // TODO 3.1.1: Заглушка, заменить  на вектор

 // double vlen = v.Length; // TODO 3.1.1: Заглушка, заменить на длину  //

DistanceUnitFormat units = (DistanceUnitFormat)4;

 // ed.WriteMessage($"Вектор 1->2: X={Converter.DistanceToString(v.X,  units, 3)},
Y={Converter.DistanceToString(v.Y, units, 3)}," +  // $"
Z={Converter.DistanceToString(v.Z, units, 3)},
длина={Converter.DistanceToString(vlen, units, 3)}");

 // // TODO 3.1.3: Использовать для вывода координат X,Y,Z и длины
Converter.DistanceToString()
 //}




 //[CommandMethod("HelloXLine")]
 //public void HelloXLine()
 //{
 // Document doc = Application.DocumentManager.MdiActiveDocument;  // Editor
ed = doc.Editor;
 // Database db = doc.Database;
 // ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать, если нужно
добавлять объекты в текущий лист, вместо ModelSpace

 // Point3d p1, p2, p3;

 // PromptPointOptions opts = new PromptPointOptions("Введите первую точку
XLINE");
 // PromptPointResult pr = ed.GetPoint(opts);

 // if (PromptStatus.OK != pr.Status)
```



```
 // return;

 // p1 = pr.Value;

 // opts.Message = "Введите вторую точку XLINE";
 // opts.BasePoint = p1;
 // opts.UseBasePoint = true;

 // pr = ed.GetPoint(opts);
 // if (PromptStatus.OK != pr.Status)
 // return;

 // p2 = pr.Value;

 // opts.Message = "Введите третью точку";
```

```
 // opts.UseBasePoint = false;

 // pr = ed.GetPoint(opts);
 // if (PromptStatus.OK != pr.Status)
 // return;

 // p3 = pr.Value;

 // using (Transaction trans =
db.TransactionManager.StartTransaction())
 // {
 // BlockTable blockTable = trans.GetObject(db.BlockTableId,
OpenMode.ForRead) as BlockTable;
 // BlockTableRecord modelSpace =
trans.GetObject(blockTable[BlockTableRecord.ModelSpace], OpenMode.ForWrite) as
BlockTableRecord;

 // // TODO 3.1.2: Создать прямую XLine по точкам p1, p2,  // // используя
xline.BasePoint, xline.SecondPoint  // // т.к. StartPoint и EndPoint не
реализованы  // Xline xline = new Xline(); // TODO 3.1.2: Создать линию  //
xline.BasePoint = p1; // BasePoint
 // xline.SecondPoint = p2; // SecondPoint

 // if (xline != null)
 // {
 // modelSpace.AppendEntity(xline);
 // trans.AddNewlyCreatedDBObject(xline, true);  // }

 // // TODO 3.1.2: Найти ближайшую к точке p3 точку прямой p3cl  // // 1. Используя
метод GetClosestPointTo() пространства имён Teigha.Geometry
 // // 2. Используя метод GetClosestPointTo() пространства имён
Teigha.DatabaseServices

 // Point3d p3cl = xline.GetClosestPointTo(p3, true);

 // Line line = new Line();
 // line.StartPoint = p3;
 // // Добавить к p3cl свойство Point (Teigha.Geometry),  // // либо
использовать p3cldb (Teigha.DatabaseServices)  // line.EndPoint = p3cl;


 // modelSpace.AppendEntity(line);
```



```
// trans.AddNewlyCreatedDBObject(line, true);

 // trans.Commit();
 // }
 //}




 //[CommandMethod("HelloIntersect")]
 //public void HelloIntersect()
 //{
 // Document doc = Application.DocumentManager.MdiActiveDocument;  // Editor
ed = doc.Editor;
 // Database db = doc.Database;
 // ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать, если нужно
добавлять объекты в текущий лист, вместо ModelSpace

 // ObjectId id1, id2;

 // PromptEntityOptions opts;
```

```
 // PromptEntityResult pr;

 // opts = new PromptEntityOptions("Выберите первый примитив");  // pr =
ed.GetEntity(opts);
 // id1 = pr.ObjectId;

 // opts.Message = "Выберите второй примитив";
 // pr = ed.GetEntity(opts);
 // id2 = pr.ObjectId;

 // using (Transaction trans =
db.TransactionManager.StartTransaction())
 // {
 // Entity ent1 = trans.GetObject(id1, OpenMode.ForRead) as Entity;  // Entity ent2 =
trans.GetObject(id2, OpenMode.ForRead) as Entity;

 // if (ent1 != null && ent2 != null)
 // {
 // ed.WriteMessage("Примитив 1: {0}\n",
ent1.GetRXClass().Name);
 // ed.WriteMessage("Примитив 2: {0}\n",
ent2.GetRXClass().Name);

 // Point3dCollection intersectionPoints = new  Point3dCollection();

 // // TODO 3.2.1: Вычислить точки пересечения примитивов ent1  и ent2,
 // // используя метод Entity.IntersectWith()  // ent1.IntersectWith(ent2,
Intersect.OnBothOperands,  intersectionPoints, 0, 0);

 // ed.WriteMessage("Число пересечений: {0}\n",
intersectionPoints.Count);

 // foreach (Point3d p in intersectionPoints)  // {
 // ed.WriteMessage("Пересечение: ({0}, {1}, {2})\n", p.X,  p.Y, p.Z);
 // }
```



```
// }
 // else
 // {
 // ed.WriteMessage("Не удалось открыть примитивы\n");  // }

 // trans.Commit();
 // }
 //}



 //[CommandMethod("HelloDistParam")]
 //public void HelloDistParam()
 //{
 // Document doc = Application.DocumentManager.MdiActiveDocument;  // Editor
ed = doc.Editor;
 // Database db = doc.Database;

 // ObjectId objectId;

 // PromptEntityOptions opts;
 // PromptEntityResult pr;

 // try
 // {
 // opts = new PromptEntityOptions("Выберите кривую");  // pr =
ed.GetEntity(opts);
 // if (pr.Status != PromptStatus.OK)
```

```
 // return;
 // objectId = pr.ObjectId;

 // using (Transaction trans =
db.TransactionManager.StartTransaction())
 // {
 // BlockTable blockTable = trans.GetObject(db.BlockTableId,  OpenMode.ForRead) as
BlockTable;
 // BlockTableRecord modelSpace =
trans.GetObject(blockTable[BlockTableRecord.ModelSpace], OpenMode.ForWrite) as
BlockTableRecord;

 // DBObject obj = trans.GetObject(objectId,  OpenMode.ForWrite);
 // Curve curve = obj as Curve;

 // if (curve != null)
 // {
 // ed.WriteMessage("Кривая: {0}\n",
curve.GetRXClass().Name);
 // ed.WriteMessage("Начальный параметр: {0}\n",  curve.StartParam);
 // ed.WriteMessage("Конечный параметр: {0}\n",  curve.EndParam);

 // double length =
curve.GetDistanceAtParameter(curve.EndParam);
 // ed.WriteMessage("Длина: {0}\n", length);

 // // Запрос количества точек
 // int numParts = 0;
 // PromptIntegerOptions intOpts = new
```



```
PromptIntegerOptions("\nВведите количество частей для разбиения кривой:");
// intOpts.AllowZero = false;
 // intOpts.AllowNegative = false;
 // PromptIntegerResult intPr = ed.GetInteger(intOpts);  // if (intPr.Status !=
PromptStatus.OK)  // return;
 // numParts = intPr.Value;

 // double step = length / numParts;

 // Point3dCollection points = new Point3dCollection();

 // // Вычисляем точки на кривой с шагом step при помощи GetPointAtDist()
 // for (double dist = 0; dist <= length; dist += step)  // {
 // Point3d point = curve.GetPointAtDist(dist);  // points.Add(point);
 // }

 // // Создаем примитивы DBPoint красного цвета  // foreach (Point3d
point in points)
 // {
 // DBPoint dbPoint = new DBPoint(point);  // dbPoint.ColorIndex = 1; //
Красный цвет  // modelSpace.AppendEntity(dbPoint);  //
trans.AddNewlyCreatedDBObject(dbPoint, true);  // }

 // // Создаем новые кривые, разбивая исходную на части  // foreach (Point3d
point in points)
 // {
 // DBObjectCollection splitCurvesColl =  curve.GetSplitCurves(new
Point3dCollection { point });
 // foreach (DBObject objCurve in splitCurvesColl)  // {
 // Curve splitCurve = objCurve as Curve;  // if (splitCurve != null)
 // {
 // modelSpace.AppendEntity(splitCurve);  //
trans.AddNewlyCreatedDBObject(splitCurve,  true);
 // }
 // }
```

```
 // }

 // // Удаляем исходную кривую
 // curve.Erase();
 // }
 // else
 // {
 // if (obj != null &&
!obj.GetRXClass().IsDerivedFrom(RXObject.GetClass(typeof(Curve))))
// {
 // ed.WriteMessage("Выбранный примитив не является  кривой\n");
 // }
 // }

 // trans.Commit();
 // }
```



```
 // }
 // catch (System.Exception ex)
 // {
 // ed.WriteMessage("Произошла ошибка: {0}", ex.Message);  // }
 //}


 ////3 модуль UCS



 //[CommandMethod("HelloRECTDIM3D")]
 //public void HelloRECTDIM3D()
 //{
 // Document doc = Application.DocumentManager.MdiActiveDocument;  // Editor
ed = doc.Editor;
 // Database db = doc.Database;
 // Matrix3d ucsMat = ed.CurrentUserCoordinateSystem;

 // // Запрос левой нижней и правой верхней точек прямоугольника  // Point3d
p1, p2;
 // Point3dCollection points = new Point3dCollection();

 // PromptPointOptions opts = new PromptPointOptions("Введите левую нижнюю точку
прямоугольника: ");
 // PromptPointResult pr = ed.GetPoint(opts);

 // if (pr.Status == PromptStatus.OK)
 // {
 // p1 = pr.Value;

 // opts.Message = "Введите правую верхнюю точку прямоугольника: ";  //
opts.BasePoint = p1;
 // opts.UseBasePoint = true;

 // pr = ed.GetPoint(opts);
 // if (pr.Status == PromptStatus.OK)
 // {
 // p2 = pr.Value;

 // // Преобразование точек из WCS в UCS
 // p1 = p1.TransformBy(ucsMat.Inverse());
 // p2 = p2.TransformBy(ucsMat.Inverse());

 // points.Add(p1);
 // points.Add(new Point3d(p1.X, p2.Y, 0));  // points.Add(p2);
 // points.Add(new Point3d(p2.X, p1.Y, 0));  // }
```

```
 // }

 // // Проверка корректности точек
 // if (points.Count != 4)
 // {
 // ed.WriteMessage("\nНеверные исходные точки\n");  // return;
 // }

 // // Создание прямоугольника
 // using (Transaction trans =
```



```
db.TransactionManager.StartTransaction())
 // {
 // BlockTable blockTable = trans.GetObject(db.BlockTableId,
OpenMode.ForRead) as BlockTable;
 // BlockTableRecord modelSpace =
trans.GetObject(blockTable[BlockTableRecord.ModelSpace], OpenMode.ForWrite) as
BlockTableRecord;

 // Polyline pline = new Polyline();
 // pline.Closed = true;

 // foreach (Point3d pt in points)
 // {
 // pline.AddVertexAt(pline.NumberOfVertices, new Point2d(pt.X,  pt.Y), 0, 0, 0);
 // }

 // // Преобразование примитива из UCS в WCS
 // pline.TransformBy(ucsMat);

 // modelSpace.AppendEntity(pline);
 // trans.AddNewlyCreatedDBObject(pline, true);

 // // Создание размеров
 // // Вертикальный размер (слева)
 // RotatedDimension dimver = new RotatedDimension();  //
dimver.XLine1Point = points[0];
 // dimver.XLine2Point = points[1];
 // dimver.DimLinePoint = new Point3d(points[0].X - 5000,  points[0].Y +
2500, 0);
 // dimver.Rotation = Math.PI / 2;
 // dimver.ColorIndex = 3; // зеленый-салатовый
 // dimver.TransformBy(ucsMat); // Преобразование примитива из UCS  в WCS
 // modelSpace.AppendEntity(dimver);
 // trans.AddNewlyCreatedDBObject(dimver, true);

 // // Горизонтальный размер (снизу)
 // RotatedDimension dimhor = new RotatedDimension();  //
dimhor.XLine1Point = points[0];
 // dimhor.XLine2Point = points[3];
 // dimhor.DimLinePoint = new Point3d(points[0].X, points[0].Y - 5000, 0);
 // dimhor.ColorIndex = 6; // розовый
 // dimhor.TransformBy(ucsMat); // Преобразование примитива из UCS  в WCS
 // modelSpace.AppendEntity(dimhor);
 // trans.AddNewlyCreatedDBObject(dimhor, true);

 // trans.Commit();
 // }
 //}



 //double m_HelloRotate3dAngleRad = Math.PI / 2;
```

```
 //[CommandMethod("HelloRotate3d")]
 //public void HelloRotate3d()
 //{
```



```
 // Document doc = Application.DocumentManager.MdiActiveDocument;  // Editor ed
= doc.Editor;
 // Database db = doc.Database;
 // ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать, если нужно
добавлять объекты в текущий лист, вместо ModelSpace

 // ObjectId axisId;

 // PromptSelectionOptions optsss;
 // PromptSelectionResult prss;

 // optsss = new PromptSelectionOptions();
 // optsss.MessageForAdding = "Выберите объекты: ";  // prss =

ed.GetSelection();

 // if (PromptStatus.OK != prss.Status)
 // return;

 // SelectionSet ss = prss.Value;

 // PromptEntityOptions opts;
 // PromptEntityResult pr;

 // opts = new PromptEntityOptions("Выберите ось вращения");  // pr =
ed.GetEntity(opts);

 // if (PromptStatus.OK != pr.Status)
 // return;

 // axisId = pr.ObjectId;

 // PromptDoubleOptions optsi = new PromptDoubleOptions("Введите угол поворота (в
градусах)");
 // optsi.DefaultValue = m_HelloRotate3dAngleRad / Math.PI * 180;  //
optsi.AllowNone = true;

 // PromptDoubleResult prd = ed.GetDouble(optsi);

 // if (PromptStatus.OK != pr.Status && PromptStatus.None != pr.Status)  // {
 // return;
 // }

 // m_HelloRotate3dAngleRad = prd.Value * Math.PI / 180;

 // using (Transaction trans =
db.TransactionManager.StartTransaction())
 // {
 // Curve axisEnt = trans.GetObject(axisId, OpenMode.ForRead) as  Curve;

 // if (axisEnt == null)
 // {
 // ed.WriteMessage("Выбранный примитив не является кривой\n");  // }

 // // TODO 3.3.3: Вычислить ось вращения и матрицу поворота  // Vector3d
axis =
axisEnt.StartPoint.GetVectorTo(axisEnt.EndPoint); // Ось вращения  // Matrix3d
mat = Matrix3d.Rotation(m_HelloRotate3dAngleRad *
```

```
Math.PI, axis, Point3d.Origin); // Матрица поворота

 // ed.WriteMessage("\nОсь вращения (X={0}, Y={1}, Z={2})  вычисляется по
примитиву: {3}\n", Converter.DistanceToString(axis.X),
Converter.DistanceToString(axis.Y), Converter.DistanceToString(axis.Z),
axisEnt.GetRXClass().Name);

 // ed.WriteMessage("Матрица преобразования:\n({1} {2} {3}  {4})\n({5} {6} {7}
{8})\n({9} {10} {11} {12})\n({13} {14} {15} {16})\n", "",  //
Converter.DistanceToString(mat[0, 0]),
Converter.DistanceToString(mat[1, 0]), Converter.DistanceToString(mat[2, 0]),
Converter.DistanceToString(mat[3, 0]),
 // Converter.DistanceToString(mat[0, 1]),
Converter.DistanceToString(mat[1, 1]), Converter.DistanceToString(mat[2, 1]),
Converter.DistanceToString(mat[3, 1]),
 // Converter.DistanceToString(mat[0, 2]),
Converter.DistanceToString(mat[1, 2]), Converter.DistanceToString(mat[2, 2]),
Converter.DistanceToString(mat[3, 2]),
 // Converter.DistanceToString(mat[0, 3]),
Converter.DistanceToString(mat[1, 3]), Converter.DistanceToString(mat[2, 3]),
Converter.DistanceToString(mat[3, 3]));

 // foreach (SelectedObject ssobj in ss)
 // {
 // ObjectId objectId = ssobj.ObjectId;
 // DBObject dbobj = trans.GetObject(objectId,  OpenMode.ForWrite);
// TODO 3.3.3 Read / Write
 // Entity ent = dbobj as Entity;

 // if (ent != null)
 // {
 // Matrix3d ecsMat = ent.Ecs;
 // ed.WriteMessage("\nПримитив {0}, матрица ECS до преобразования:\n({1}
{2} {3} {4})\n({5} {6} {7} {8})\n({9} {10} {11}  {12})\n({13} {14} {15}
{16})\n", ent.GetRXClass().Name,
 // Converter.DistanceToString(ecsMat[0, 0]),  Converter.DistanceToString(ecsMat[1,
0]), Converter.DistanceToString(ecsMat[2,  0]),
Converter.DistanceToString(ecsMat[3, 0]),
 // Converter.DistanceToString(ecsMat[0, 1]),  Converter.DistanceToString(ecsMat[1,
1]), Converter.DistanceToString(ecsMat[2,  1]),
Converter.DistanceToString(ecsMat[3, 1]),
 // Converter.DistanceToString(ecsMat[0, 2]),  Converter.DistanceToString(ecsMat[1,
2]), Converter.DistanceToString(ecsMat[2,  2]),
Converter.DistanceToString(ecsMat[3, 2]),
 // Converter.DistanceToString(ecsMat[0, 3]),  Converter.DistanceToString(ecsMat[1,
3]), Converter.DistanceToString(ecsMat[2,  3]),
Converter.DistanceToString(ecsMat[3, 3]));

 // // TODO 3.3.3: Преобразовать примитив с помощью  TransformBy()

 // ent.TransformBy(mat);
 // // TODO: Бонус. Привести к примитиву, имеющему свойство  Normal, вывести нормаль

 // ecsMat = ent.Ecs;
 // ed.WriteMessage("Матрица ECS после
преобразования:\n({1} {2} {3} {4})\n({5} {6} {7} {8})\n({9} {10} {11}
{12})\n({13} {14} {15} {16})\n", "",
 // Converter.DistanceToString(ecsMat[0, 0]),  Converter.DistanceToString(ecsMat[1,
0]), Converter.DistanceToString(ecsMat[2,
```



```
0]), Converter.DistanceToString(ecsMat[3, 0]),
 // Converter.DistanceToString(ecsMat[0, 1]),  Converter.DistanceToString(ecsMat[1,
1]), Converter.DistanceToString(ecsMat[2,  1]),
Converter.DistanceToString(ecsMat[3, 1]),
 // Converter.DistanceToString(ecsMat[0, 2]),  Converter.DistanceToString(ecsMat[1,
```

```
2]), Converter.DistanceToString(ecsMat[2,  2]),
Converter.DistanceToString(ecsMat[3, 2]),
 // Converter.DistanceToString(ecsMat[0, 3]),  Converter.DistanceToString(ecsMat[1,
3]), Converter.DistanceToString(ecsMat[2,  3]),
Converter.DistanceToString(ecsMat[3, 3]));
 // }
 // }

 // trans.Commit();
 // }
 //}




 //[CommandMethod("HelloAddPolyVertex")]
 //public void HelloAddPolyVertex()
 //{
 // Document doc = Application.DocumentManager.MdiActiveDocument;  // Editor
ed = doc.Editor;
 // Database db = doc.Database;
 // ObjectId CurrentSpaceId = db.CurrentSpaceId;

 // ObjectId plineId;

 // // Запрос у пользователя полилинии
 // PromptEntityOptions opts = new PromptEntityOptions("\nВыберите полилинию:
");
 // opts.SetRejectMessage("\nВыбранный объект не является
полилинией.");
 // opts.AddAllowedClass(typeof(Polyline), false);

 // PromptEntityResult pr = ed.GetEntity(opts);
 // if (pr.Status != PromptStatus.OK)
 // return;

 // // Получение ID полилинии
 // plineId = pr.ObjectId;

 // // Получение плоскости полилинии и создание объекта Plane  // using
(Transaction trans =
db.TransactionManager.StartTransaction())
 // {
 // Polyline pline = trans.GetObject(plineId, OpenMode.ForRead) as  Polyline;
 // if (pline == null)
 // return;

 // // Получение координат точек для создания плоскости  // Point3d p1 =
pline.GetPoint3dAt(0);
 // Point3d p2 = pline.GetPoint3dAt(1);
 // Point3d p3 = pline.GetPoint3dAt(2);


 // ed.WriteMessage("\nxyz1 {0} {1} {2}", p1.X, p1.Y, p1.Z);  //
ed.WriteMessage("\nxyz2 {0} {1} {2}", p2.X, p2.Y, p2.Z);
```



```
 // ed.WriteMessage("\nxyz3 {0} {1} {2}", p3.X, p3.Y, p3.Z);  // // Создание
плоскости через три точки
 // Plane plane = new Plane(p1, p2, p3);

 // // Запрос у пользователя точки
 // PromptPointOptions ppo = new PromptPointOptions("\nВыберите точку: ");
 // PromptPointResult ppr = ed.GetPoint(ppo);
 // if (ppr.Status != PromptStatus.OK)
```

```
 // return;

 // Point3d point = ppr.Value;
 // ed.WriteMessage("\nxyzpoint {0} {1} {2}", point.X, point.Y,  point.Z);
 // // Проверка, лежит ли точка в плоскости полилинии  // double distance =
plane.GetSignedDistanceTo(point);  // ed.WriteMessage("\ndist {0}",
Math.Abs(distance));  // ed.WriteMessage("\ntolerans {0}",
Tolerance.Global.EqualPoint);  // if (Math.Abs(distance) >
Tolerance.Global.EqualPoint)  // {
 // ed.WriteMessage("\nВыбранная точка не лежит в плоскости  полилинии.");
 // return;
 // }

 // // Открытие полилинии на запись
 // pline.UpgradeOpen();
 // }
 //}



 //4 модуль Props



 //[CommandMethod("HelloDict")]
 //public void HelloDict()
 //{
 // Document doc = Application.DocumentManager.MdiActiveDocument;  // Editor
ed = doc.Editor;
 // Database db = doc.Database;
 // ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать, если нужно
добавлять объекты в текущий лист, вместо ModelSpace

 // using (Transaction trans =
db.TransactionManager.StartTransaction())
 // {
 // #region TODO 4.1.1:

 // ed.WriteMessage("\nNamed objects dictionary");

 // // Сохранение ObjectId словаря словарей в переменную dictId  // ObjectId
dictId = db.NamedObjectsDictionaryId;

 // // TODO 4.1.1.A: Открыть словарь словарей для чтения  // DBDictionary dict =
trans.GetObject(dictId, OpenMode.ForRead)  as DBDictionary;

 // // TODO 4.1.1.B: перебрать содержимое словаря при помощи цикла:  // foreach
(DBDictionaryEntry dictEntry in dict)
```



```
 // {
 // String key = dictEntry.Key;
 // DBObject value = trans.GetObject(dictEntry.Value,  OpenMode.ForRead) as
DBObject;
 // string valueString = value.GetRXClass().Name;

 // ed.WriteMessage("\n Имя: {0}, Тип: {1} / {2}", key,  value.GetType(),
valueString);
 // }

 // // TODO 4.1.1.C: во время перебора вывести в командную строку  // // 1. имя
каждого вхождения (свойство key)
 // // 2. тип каждого вхождения, который можно узнать только открыв  это
 // // вхождение в базе данных (для этого берем свойство value)  // // как
DBObject и воспользовавшись методом GetType() или  GetRXClass().Name
 // #endregion
```

```
 // #region TODO 4.1.2:

 // ed.WriteMessage("\n\nLayout dictionary");

 // string dictName = "ACAD_LAYOUT";
 // ObjectId layout = dict.GetAt(dictName);

 // // TODO 4.1.2.A: Получить ObjectId словаря ACAD_LAYOUT методом  // //
DBDictionary.GetAt(string dictName) либо через класс Database:
 // // db.LayoutDictionaryId
 // DBDictionary layout_dict = trans.GetObject(layout,
OpenMode.ForRead) as DBDictionary;
 // // TODO 4.1.2.B: Открыть словарь листов для чтения  // foreach
(DBDictionaryEntry layout_dictEntry in layout_dict)  // {
 // ed.WriteMessage("\nkey = {0}", layout_dictEntry.Key);  // }
 // // TODO 4.1.2.C: Перебрать в цикле содержимое словаря,  // // выводя в командную
строку названия вхождений (свойство key)

 // #endregion

 // trans.Commit();
 // }
 //}



 //[CommandMethod("HelloChProps")]
 //public void HelloChProps()
 //{
 // Document doc = Application.DocumentManager.MdiActiveDocument;  // Editor
ed = doc.Editor;
 // Database db = doc.Database;
 // ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать, если нужно
добавлять объекты в текущий лист, вместо ModelSpace

 // using (Transaction trans =
db.TransactionManager.StartTransaction())
 // {
 // // Выбор примитива
```



```
 // PromptEntityOptions opts;
 // PromptEntityResult pr;

 // // Запрос пользователю
 // opts = new PromptEntityOptions("Выберите примитив");  // pr =
ed.GetEntity(opts);

 // if (PromptStatus.OK != pr.Status)
 // return;

 // // Сохранение ObjectId выбранного примитива  // ObjectId
entityId = pr.ObjectId;


 // // Словарь стилей печати
 // ed.WriteMessage("\n\nPlot style name dictionary");

 // // TODO: Сохранить ObjectId словаря стилей в переменную
plotStyleNameDictId
 // ObjectId plotStyleNameDictId = db.PlotStyleNameDictionaryId;

 // // !!! В словаре стилей печати хранятся только используемые в  документе стили
 // // Если стили печати никак не используются, то по умолчанию в  этом словаре
только стиль Normal
```

```
 // // TODO: Открыть словарь стилей печати для записи, чтобы  добавить новый
стиль печати
 // DBDictionary plotStyleNameDict =
trans.GetObject(plotStyleNameDictId, OpenMode.ForWrite) as DBDictionary;

 // // Создание нового стиля печати для словаря стилей печати  // PlaceHolder
placeHolder = new PlaceHolder();  // string newStyleName = "HelloPlotStyle";
 // plotStyleNameDict.SetAt(newStyleName, placeHolder);  // // TODO:
Добавить новый стиль в словарь стилей печати  // // методом
DBDictionary.SetAt(),
 // // используя в качестве ключа переменную newStyleName,  // // в
качестве value переменную placeHolder

 // foreach (DBDictionaryEntry entry in plotStyleNameDict)  // {
 // ed.WriteMessage("\nkey - {0}", entry.Key);  // }

 // // TODO: Перебрать циклом содержимое словаря стилей печати,  // // выводя
названия вхождений словаря в командную строку  // //foreach(DBDictionaryEntry
entry in plotStyleNameDict)


 // // Опции запроса нового стиля печати
 // PromptStringOptions sopts;
 // PromptResult spr;
 // sopts = new PromptStringOptions("\nВведите имя нового стиля  печати
примитива");
 // sopts.DefaultValue = "HelloPlotStyle";
 // // Запрос нового стиля печати
 // spr = ed.GetString(sopts);

 // if (PromptStatus.OK != spr.Status)
 // return;
```



```
 // // Сохранение имени нового стиля печати
 // String newPlotStyleName = spr.StringResult;

 // // TODO: Сохранить ObjectId нового стиля печати в переменную   // // ObjectId
нужного вхождения словаря можно найти методом  DBDictionary.GetAt(string name)
 // ObjectId newPlotStyleNameId =
plotStyleNameDict.GetAt(newPlotStyleName);


 // if (newPlotStyleNameId == ObjectId.Null)
 // {
 // ed.WriteMessage("\nСтиль печати {0} не найден",  newPlotStyleName);
 // return;
 // }


 // // Таблица слоёв

 // // Создание нового слоя
 // LayerTableRecord tableRecord = new LayerTableRecord();  // sopts = new
PromptStringOptions("\nВведите имя нового слоя");  // sopts.AllowSpaces = true;
 // spr = ed.GetString(sopts);
 // tableRecord.Name = spr.StringResult; /* Имя нового слоя*/  //
tableRecord.Color =
Color.FromColor(System.Drawing.Color.BurlyWood); /*Цвет*/
 // tableRecord.LineWeight = LineWeight.LineWeight040; /*Толщина линий слоя*/
 // tableRecord.PlotStyleNameId = newPlotStyleNameId; /*Стиль печати слоя*/

 // // Добавление нового слоя в таблицу слоёв
 // // TODO : Открыть таблицу слоёв для записи  // LayerTable layerTable =
trans.GetObject(db.LayerTableId,  OpenMode.ForWrite) as LayerTable;
```

```
// foreach (ObjectId entry in layerTable)
// {
// bool v = entry.Compare(newPlotStyleNameId);  // if (v)
// {
// ed.WriteMessage("---");
// }
// }
// // TODO : Выполнить проверку, нет ли в ней слоя с тем же  именем,
// // что у созданного нами слоя. Если нет, то добавляем новый  слой
// // в таблицу методом LayerTable.Add()
// layerTable.Add(tableRecord);

// // Вывод списка слоёв в командную строку
// foreach (ObjectId layerId in layerTable)
// {
// DBObject obj = trans.GetObject(layerId, OpenMode.ForRead,  false, true);
// LayerTableRecord ltr = obj as LayerTableRecord;
```



```
// if (ltr != null)
// {
// string layerName = ltr.Name;
// long handle = ltr.Handle.Value;

// ed.WriteMessage("\nСлой: {0}, handle={1}", layerName,  handle);
// }
// }

// // Выбор нового слоя примитива
// sopts = new PromptStringOptions("\nВведите имя слоя, на который  переместить
примитив");
// sopts.AllowSpaces = true;
// spr = ed.GetString(sopts);

// if (PromptStatus.OK != spr.Status)
// return;

// // Сохранение имени нового слоя
// String newLayerName = spr.StringResult;

// // TODO : Сохранить ObjectId выбранного слоя в переменную newLayerId
// // ObjectId слоя можно найти в таблице слоёв методом
LayerTable[newLayerName]
// ObjectId newLayerId = layerTable[newLayerName];


// if (newLayerId == ObjectId.Null)
// {
// ed.WriteMessage("\nСлой {0} не найден", newLayerName);  // return;
// }


// //TODO 4.1.3.C: Изменение свойств примитива

// // Установка свойств примитива
// // TODO: Открыть примитив для записи
// Entity prim = trans.GetObject(entityId, OpenMode.ForWrite) as  Entity;

// if (prim != null)
// {
// prim.PlotStyleNameId = new
PlotStyleDescriptor(newPlotStyleNameId, PlotStyleNameType.PlotStyleNameById);
// prim.LayerId = newLayerId;
// }
```

```
 // // Стиль печати
 // // TODO : Установить новый стиль печати (new
PlotStyleDescriptor(newPlotStyleNameId, PlotStyleNameType.PlotStyleNameById)),   //
// изменив свойство PlotStyleNameId


 // // Слой
 // // TODO : Установить новый слой, изменив свойство LayerId
```



```
 // trans.Commit();
 // }
 //}

 ////4 модуль Jig

 //public class RectEntJig : EntityJig
 //{
 // // Переменные для точек, вокруг которых будет отрисовываться
прямоугольник
 // public Point3d basePt, newPt;

 // // Конструктор класса, в котором обязательный параметр - примитив.  // // Этот
примитив записывается в свойство Entity класса EntityJig  // public
RectEntJig(Polyline rect, Point3d pt)
 // : base(rect)
 // {
 // this.basePt = pt;
 // this.newPt = pt;
 // }

 // // В методе обрабатываются данные о перемещении курсора мыши  //
protected override SamplerStatus Sampler(JigPrompts jp)  // {
 // // Запросить правую верхнюю (вторую) точку прямоугольника при  помощи
jp.AcquirePoint()
 // PromptPointResult ppr = jp.AcquirePoint("Введите вторую точку  прямоугольника:
");

 // // Если точка получена (PromptStatus.OK), то  // if
(ppr.Status == PromptStatus.OK)
 // {
 // // Если вторая точка (ppr.Value) совпадает с первой  (basePt), то
возвращаем SamplerStatus.NoChange,
 // // иначе получится вырожденная полилиния с совпадающими  точками
 // if (ppr.Value.IsEqualTo(basePt))
 // {
 // return SamplerStatus.NoChange;
 // }
 // else
 // {
 // // Если точка newPt не совпадает с basePt, то обновляем точку newPt и возвращаем
SamplerStatus.OK
 // newPt = new Point3d(ppr.Value.X, ppr.Value.Y, 0);  // return
SamplerStatus.OK;
 // }
 // }
 // else
 // {
 // // Если точка не получена, возвращаем SamplerStatus.Cancel  // return
SamplerStatus.Cancel;
 // }
 // }
```

```
 // // В методе изменяются параметры примитива с использованием данных,  // //
полученных в методе Sampler()
 // protected override bool Update()
 // {
 // // Для отрисовки предпросмотра используем свойство Entity,  // // в которое
запомнился примитив, который мы передали в
```



```
конструктор
 // Polyline pline = Entity as Polyline;

 // if (pline == null)
 // {
 // return false;
 // }

 // // Добавляем вершины полилинии так, чтобы получился  прямоугольник
 // if (pline.NumberOfVertices == 0)
 // {
 // // Добавить в пустую полилинию четыре угловых точки  // pline.AddVertexAt(0, new
Point2d(basePt.X, basePt.Y), 0, 0,  0);
 // pline.AddVertexAt(1, new Point2d(newPt.X, basePt.Y), 0, 0,  0);
 // pline.AddVertexAt(2, new Point2d(newPt.X, newPt.Y), 0, 0,  0);
 // pline.AddVertexAt(3, new Point2d(basePt.X, newPt.Y), 0, 0,  0);
 // pline.Closed = true;
 // }
 // else
 // {
 // // Обновить точки полилинии, установить правый верхний угол  в newPt
 // pline.SetPointAt(1, new Point2d(newPt.X, basePt.Y));  //
pline.SetPointAt(2, new Point2d(newPt.X, newPt.Y));  // pline.SetPointAt(3,
new Point2d(basePt.X, newPt.Y));  // }

 // return true;
 // }
 //}

 //public class RectDrawJig : DrawJig
 //{
 // // Переменные для точек, вокруг которых будут отрисовываться
прямоугольник
 // public Point3d basePt, newPt;

 // // Примитив, которым будет отрисовываться прямоугольник  // public
Polyline pline;

 // // Горизонтальный и вертикальный размеры
 // public RotatedDimension dimhor, dimver;

 // // Конструктор класса без обязательных параметров  // public
RectDrawJig(Point3d basePt)
 // {
 // this.basePt = basePt;
 // this.newPt = basePt;
 // }

 // protected override SamplerStatus Sampler(JigPrompts jp)  // {
 // // Запросить правую верхнюю (вторую) точку прямоугольника при  помощи
jp.AcquirePoint()
 // PromptPointResult ppr = jp.AcquirePoint("\nУкажите вторую точку  прямоугольника:
");
```



```
 // // Если точка получена (PromptStatus.OK), то  // if
(ppr.Status == PromptStatus.OK)
```

```
 // {
 // // Если вторая точка (ppr.Value) совпадает с первой  (basePt), то
возвращаем SamplerStatus.NoChange,
 // // иначе получится вырожденная полилиния с совпадающими  точками
 // if (ppr.Value.IsEqualTo(basePt))
 // {
 // return SamplerStatus.NoChange;
 // }
 // else
 // {
 // // Если точка newPt не совпадает с basePt, то обновляем точку newPt и возвращаем
SamplerStatus.OK
 // newPt = new Point3d(ppr.Value.X, ppr.Value.Y, 0);  // return
SamplerStatus.OK;
 // }
 // }
 // else
 // {
 // // Если точка не получена, возвращаем SamplerStatus.Cancel  // return
SamplerStatus.Cancel;
 // }
 // }

 // protected override bool
WorldDraw(PlatformDb.GraphicsInterface.WorldDraw draw)
 // {
 // if (pline == null)
 // {
 // pline = new Polyline();
 // pline.SetDatabaseDefaults();
 // }

 // // Добавляем вершины в полилинию так, чтобы получился  прямоугольник
 // if (pline.NumberOfVertices == 0)
 // {
 // // Добавить в пустую полилинию четыре угловых точки  // pline.AddVertexAt(0, new
Point2d(basePt.X, basePt.Y), 0, 0,  0);
 // pline.AddVertexAt(1, new Point2d(basePt.X, newPt.Y), 0, 0,  0);
 // pline.AddVertexAt(2, new Point2d(newPt.X, newPt.Y), 0, 0,  0);
 // pline.AddVertexAt(3, new Point2d(newPt.X, basePt.Y), 0, 0,  0);
 // pline.Closed = true;
 // }
 // else
 // {
 // // Обновить точки полилинии, установить правый верхний угол  в newPt
 // pline.SetPointAt(1, new Point2d(basePt.X, newPt.Y));  //
pline.SetPointAt(2, new Point2d(newPt.X, newPt.Y));  // pline.SetPointAt(3,
new Point2d(newPt.X, basePt.Y));  // }

 // // Отрисовка предпросмотра полилинии – прямоугольника
```



```
 // draw.Geometry.Draw(pline);

 // // Коллекция точек – вершин полилинии
 // Point3dCollection points = new Point3dCollection();

 // points.Add(new Point3d(basePt.X, basePt.Y, 0));  //
points.Add(new Point3d(basePt.X, newPt.Y, 0));  // points.Add(new
Point3d(newPt.X, newPt.Y, 0));  // points.Add(new Point3d(newPt.X,
basePt.Y, 0));

 // if (basePt.DistanceTo(newPt) > 5000)
 // {
 // // Диагональ – отрисовка геометрией
 // // Выставить красный цвет (1) при помощи
```

```
draw.SubEntityTraits.Color и нарисовать диагональ при помощи
draw.Geometry.WorldLine()
 // draw.SubEntityTraits.Color = 1;
 // draw.Geometry.WorldLine(points[0], points[2]);

 // // Вертикальный размер
 // if (dimver == null)
 // {
 // dimver = new RotatedDimension();
 // dimver.SetDatabaseDefaults();
 // dimver.ColorIndex = 3;
 // }

 // // Обновить положение вертикального размера (XLine1Point,  XLine2Point,
DimLinePoint, Rotation)
 // dimver.XLine1Point = new Point3d(newPt.X, basePt.Y, 0);  // dimver.XLine2Point =
new Point3d(newPt.X, newPt.Y, 0);  // dimver.DimLinePoint = new Point3d(newPt.X +
100, (basePt.Y  + newPt.Y) / 2, 0); // Примерное положение текста размера
 // dimver.Rotation = Math.PI / 2; // Поворот размерной линии  на 90 градусов
 // dimver.RecomputeDimensionBlock(true); // Пересчитать  служебный блок
размера
 // draw.Geometry.Draw(dimver); // Отрисовать размер - отрисовка
примитивами

 // // Горизонтальный размер
 // if (dimhor == null)
 // {
 // dimhor = new RotatedDimension();
 // dimhor.SetDatabaseDefaults();
 // dimhor.ColorIndex = 3;
 // }

 // // Обновить положение горизонтального размера (XLine1Point,  XLine2Point,
DimLinePoint)
 // dimhor.XLine1Point = new Point3d(basePt.X, newPt.Y, 0);  // dimhor.XLine2Point =
new Point3d(newPt.X, newPt.Y, 0);  // dimhor.DimLinePoint = new Point3d((basePt.X +
newPt.X) / 2,  newPt.Y + 100, 0); // Примерное положение текста размера
 // dimhor.Rotation = 0; // Поворот размерной линии на 0  градусов
 // dimhor.RecomputeDimensionBlock(true); // Пересчитать  служебный блок
размера
 // draw.Geometry.Draw(dimhor); // Отрисовать размер - отрисовка
примитивами
 // }
```



```
 // return true;
 // }
 //}

 //public partial class Commands
 //{
 // // TODO 4.2.1.D: Изучить команду HelloEntJig
 // [CommandMethod("HelloEntJig")]
 // public void HelloRECTDIMEntJig()
 // {
 // // Ссылки на активный dwg-документ, его редактор и базу данных  // Document doc
= Application.DocumentManager.MdiActiveDocument;  // Editor ed = doc.Editor;
 // Database db = doc.Database;
 // ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать,  если нужно
добавлять объекты в текущий лист, вместо ModelSpace

 // // Переменная для базовой точки прямоугольника  // Point3d p1;

 // // Запрос базовой точки прямоугольника
 // PromptPointOptions opts = new PromptPointOptions("Левая нижняя точка
прямоугольника: ");
```

```
 // PromptPointResult pr = ed.GetPoint(opts);

 // if (PromptStatus.OK != pr.Status)
 // return;

 // // Сохранение координат базовой точки в переменную  // p1 =
pr.Value;

 // // Начало работы с базой данных документа
 // using (Transaction trans =
db.TransactionManager.StartTransaction())
 // {
 // // Открываем таблицу блоков
 // BlockTable blockTable = trans.GetObject(db.BlockTableId,  OpenMode.ForRead) as
BlockTable;

 // // Открываем пространство модели из таблицы блоков  // BlockTableRecord
modelSpace =
trans.GetObject(blockTable[BlockTableRecord.ModelSpace], OpenMode.ForWrite) as
BlockTableRecord;

 // // Создание примитива, с помощью которого будет  отрисовываться
прямоугольник
 // Polyline pline = new Polyline();

 // // Создание экземпляра класса, отрисовывающего  прямоугольник с
технологией Jig
 // RectEntJig rectJig = new RectEntJig(pline, p1);

 // // Запускаем отрисовку примитива с технологией Jig (см.  класс RectEntJig)
 // pr = (PromptPointResult)ed.Drag(rectJig);  // if (pr.Status ==
PromptStatus.OK)
 // {

 // // Добавляем созданный примитив в пространство модели  //
modelSpace.AppendEntity(pline);
```



```
 // // Добавляем созданный примитив в базу данных чертежа  //
trans.AddNewlyCreatedDBObject(pline, true);

 // // Завершаем транзакцию
 // trans.Commit();
 // }
 // }
 // }

 // [CommandMethod("HelloDrawJig")]
 // public void HelloRECTDIMDrawJig()
 // {
 // // Ссылки на активный dwg-документ, его редактор и базу данных  // Document doc
= Application.DocumentManager.MdiActiveDocument;  // Editor ed = doc.Editor;
 // Database db = doc.Database;
 // ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать,  если нужно
добавлять объекты в текущий лист, вместо ModelSpace

 // // Переменная для базовой точки прямоугольника  // Point3d p1;

 // // Запрос базовой точки прямоугольника
 // PromptPointOptions opts = new PromptPointOptions("Левая нижняя точка
прямоугольника: ");
 // PromptPointResult pr = ed.GetPoint(opts);

 // if (PromptStatus.OK != pr.Status)
 // return;
```

```
 // // Сохранение координат базовой точки в переменную  // p1 =
pr.Value;

 // // Начало транзакции с базой данных чертежа  // using
(Transaction trans =
db.TransactionManager.StartTransaction())
 // {
 // // Открываем таблицу блоков чертежа
 // BlockTable blockTable = trans.GetObject(db.BlockTableId,  OpenMode.ForRead) as
BlockTable;

 // // Открываем пространство модели из таблицы блоков  // BlockTableRecord
modelSpace =
trans.GetObject(blockTable[BlockTableRecord.ModelSpace], OpenMode.ForWrite) as
BlockTableRecord;

 // // Создаем экземпляр класса, который отрисовываем  прямоугольник с
 // // размерами по двум сторонам с применением технологии Jig  // RectDrawJig
rectJig = new RectDrawJig(p1);

 // // Запускаем отрисовку произвольной графики с технологией  Jig
 // pr = (PromptPointResult)ed.Drag(rectJig);  // if (pr.Status ==
PromptStatus.OK)
 // {
 // // TODO 4.2.2.G: Добавить rectJig.pline,  rectJig.dimver,
rectJig.dimhor в чертёж

 // // Добавляем полилинию-прямоуголь
```



```
 // // Добавляем полилинию-прямоугольник в пространство  модели и базу данных
чертежа
 // modelSpace.AppendEntity(rectJig.pline);  //
trans.AddNewlyCreatedDBObject(rectJig.pline, true);

 // // Добавляем вертикальный размер в пространство модели  и базу данных чертежа
 // modelSpace.AppendEntity(rectJig.dimver);  //
trans.AddNewlyCreatedDBObject(rectJig.dimver, true);

 // // Добавляем горизонтальный размер в пространство  модели и базу данных
чертежа
 // modelSpace.AppendEntity(rectJig.dimhor);  //
trans.AddNewlyCreatedDBObject(rectJig.dimhor, true);

 // // Завершаем транзакцию
 // trans.Commit();
 // }
 // }
 // }
 //}


 //5 модуль


 //TODO 5.3.1: Добавить наследование от IExtensionApplication

 public partial class Commands : IExtensionApplication
 {
 ObjectId m_LineId;
 ObjectId old_n_LineId;
 bool remind = false;
```

```
 void HelloDbReactor_ObjectAppended(object sender, ObjectEventArgs e)  {
 }

 void HelloDbReactor_ObjectErased(object sender, ObjectErasedEventArgs  e)
 {
 }

 void HelloDbReactor_ObjectModified(object sender, ObjectEventArgs e)  {

 // TODO 5.1.2: Проверить, m_LineId ли изменён,  // если да, вывести
сообщение "Отрезок изменён"  // и начальную и конечную точки
 if (e.DBObject.ObjectId == m_LineId)
 {
       Document doc = Application.DocumentManager.MdiActiveDocument; Editor ed =
                                      doc.Editor;
                     Database db = doc.Database;

 using (Transaction trans =
db.TransactionManager.StartTransaction())
 {
 Line line = trans.GetObject(m_LineId, OpenMode.ForRead)
```



```
as Line;
 if (line != null)
                          {
 Point3d p1, p2;

 p1 = line.StartPoint;
                               p2 = line.EndPoint;

 ed.WriteMessage("\nОтрезок изменён\n");  ed.WriteMessage("Начальная точка: ({0}, {1},
{2})\n",  p1.X, p1.Y, p1.Z);
 ed.WriteMessage("Конечная точка: ({0}, {1}, {2})\n",  p2.X, p2.Y, p2.Z);
 }

 trans.Commit();
 }

 }

 }

 void HelloDbReactor_ObjectOpenedForModify(object sender,
ObjectEventArgs e)
 {
 }

 void HelloDbReactor_ObjectReappended(object sender, ObjectEventArgs  e)
 {
 }

 void HelloDbReactor_ObjectUnappended(object sender, ObjectEventArgs  e)
 {
 Line line = sender as Line;

 if (line != null)
 {
 Point3d p1, p2;

 p1 = line.StartPoint;
                     p2 = line.EndPoint;

       Document doc = Application.DocumentManager.MdiActiveDocument; Editor ed =
                                      doc.Editor;
```

```
 ed.WriteMessage("\nОтрезок изменён\n");
 ed.WriteMessage("Начальная точка: ({0}, {1}, {2})\n", p1.X,  p1.Y, p1.Z);
 ed.WriteMessage("Конечная точка: ({0}, {1}, {2})\n", p2.X,  p2.Y, p2.Z);
 }

 }

 void HelloDbReactor_SystemVariableWillChange(object sender,
PlatformDb.DatabaseServices.SystemVariableChangingEventArgs e)  {
 // TODO 5.1.3: вывести в командную строку название и старое  значение
системной переменной
```



```
 Document doc = Application.DocumentManager.MdiActiveDocument;  Editor ed =
doc.Editor;
 string value =
System.Convert.ToString(Application.GetSystemVariable(e.Name));
ed.WriteMessage($"Название: {e.Name}, Значение:{value}\n");

 }

 void HelloDbReactor_SystemVariableChanged(object sender,
PlatformDb.DatabaseServices.SystemVariableChangedEventArgs e)  {
 // TODO 5.1.3: вывести в командную строку название и новое  значение
системной переменной

 Document doc = Application.DocumentManager.MdiActiveDocument;  Editor ed =
doc.Editor;
 Database db = sender as Database;
 string value =
System.Convert.ToString(Application.GetSystemVariable(e.Name));
ed.WriteMessage($"Название: {e.Name}, Значение:{value}\n");  }

 [CommandMethod("HelloDbReactorAdd")]
 public void HelloDbReactorAdd()
 {
 Document doc = Application.DocumentManager.MdiActiveDocument;  Editor ed =
doc.Editor;
 Database db = doc.Database;
 ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать,  если нужно
добавлять объекты в текущий лист, вместо ModelSpace


 PromptEntityOptions entOpt = new PromptEntityOptions("Выберите отрезок: ");
 PromptEntityResult entRes = ed.GetEntity(entOpt);

 var entId = entRes.ObjectId;
 bool isLine = false;

 using (Transaction trans =
db.TransactionManager.StartTransaction())
 {
 Entity entity = trans.GetObject(entId, OpenMode.ForRead) as  Entity;

 if (entity.GetType() == typeof(Line))
                     {
 isLine = true;
 }
                     trans.Commit();

 }

 if (isLine)
 {
 m_LineId = entId;
```

```
 if (m_LineId != ObjectId.Null)
                     {
```



```
 ed.WriteMessage("\nРеакторы уже установлены. Снимите реакторы командой
HelloDbReactorRemove");
 //return;
 }

 PromptEntityOptions entOpt2 = new
PromptEntityOptions("Выберите отрезок: ");
 PromptEntityResult entRes2 = ed.GetEntity(entOpt2);

 var entId2 = entRes.ObjectId;
                     bool isLine2 = false;

 using (Transaction trans =
db.TransactionManager.StartTransaction())
 {
 Entity entity = trans.GetObject(entId, OpenMode.ForRead)  as Entity;

             if (entity.GetType() == typeof(Line)) {
 isLine2 = true;
 }
                         trans.Commit();
 }

 if (isLine2)
                     {
 m_LineId = entId;
 }

 // TODO 5.1.2: Выбрать отрезок, запомнить его ObjectId в m_LineId

 // TODO 5.1.1: Подписаться на события, используя заготовки  HelloDbReactor*:
 // ObjectAppended, ObjectOpenedForModify, ObjectReappended,  ObjectUnappended

 db.SystemVariableWillChange +=
HelloDbReactor_SystemVariableWillChange;
 db.SystemVariableChanged +=
HelloDbReactor_SystemVariableChanged;

 db.ObjectErased += HelloDbReactor_ObjectErased;  db.ObjectModified +=

HelloDbReactor_ObjectModified;

 db.ObjectReappended += HelloDbReactor_ObjectReappended; db.ObjectUnappended
                     += HelloDbReactor_ObjectUnappended;

 db.ObjectAppended += HelloDbReactor_ObjectAppended;  db.ObjectAppended +=

HelloDbReactor_ObjectOpenedForModify;


 }
 else
 {
 ed.WriteMessage("Увы, но вы выбрали либо не отрезок, либо  ничего не выбрали
:(");
 }
```



```
}
```

```
 [CommandMethod("HelloDbReactorRemove")]
 public void HelloDbReactorRemove()
 {
 Document doc = Application.DocumentManager.MdiActiveDocument;  Editor ed =
doc.Editor;
 Database db = doc.Database;
 ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать,  если нужно
добавлять объекты в текущий лист, вместо ModelSpace

 m_LineId = ObjectId.Null;
 PromptEntityOptions entOpt = new PromptEntityOptions("Выберите отрезок: ");
 PromptEntityResult entRes = ed.GetEntity(entOpt);

 var entId = entRes.ObjectId;
 bool isLine = false;

 using (Transaction trans =
db.TransactionManager.StartTransaction())
 {
 Entity entity = trans.GetObject(entId, OpenMode.ForRead) as  Entity;

 if (entity.GetType() == typeof(Line))
                     {
 isLine = true;
 }
                     trans.Commit();
 }

 if (isLine)
 {
 m_LineId = entId;

 if (m_LineId != ObjectId.Null)
                     {
 ed.WriteMessage("\nРеакторы уже сняты. Установите  реакторы с помощью
HelloDbReactorAdd\n");
 return;
 }

 db.SystemVariableWillChange -=
HelloDbReactor_SystemVariableWillChange;
 db.SystemVariableChanged -=
HelloDbReactor_SystemVariableChanged;

 db.ObjectErased -= HelloDbReactor_ObjectErased;  db.ObjectModified -=

HelloDbReactor_ObjectModified;

 db.ObjectReappended -= HelloDbReactor_ObjectReappended; db.ObjectUnappended -
                     = HelloDbReactor_ObjectUnappended;

 db.ObjectAppended -= HelloDbReactor_ObjectAppended;  db.ObjectAppended -=

HelloDbReactor_ObjectOpenedForModify;
```



```
 }
 else
 {
 ed.WriteMessage("Вы выбрали не отрезок, либо ничего не  выбрали.");
 }
 // TODO 5.1.1: Отписаться от событий
 // ObjectAppended, ObjectErased, ObjectModified,
ObjectOpenedForModify, ObjectReappended, ObjectUnappended
```

```
 }

 void HelloObjectReactor_Erased(object sender, ObjectErasedEventArgs  e)
 {
 }

 void HelloObjectReactor_Modified(object sender, EventArgs e)  {
 Line line = sender as Line;

 if (line != null)
 {
 Point3d p1, p2;

 p1 = line.StartPoint;
                     p2 = line.EndPoint;

       Document doc = Application.DocumentManager.MdiActiveDocument; Editor ed =
                                     doc.Editor;

 ed.WriteMessage("\nОтрезок изменён\n");
 ed.WriteMessage("Начальная точка: ({0}, {1}, {2})\n", p1.X,  p1.Y, p1.Z);
 ed.WriteMessage("Конечная точка: ({0}, {1}, {2})\n", p2.X,  p2.Y, p2.Z);
 }
 }

 void HelloObjectReactor_OpenedForModify(object sender, EventArgs e)  {
 }

 void HelloObjectReactor_Reappended(object sender, EventArgs e)  {
 }

 void HelloObjectReactor_Unappended(object sender, EventArgs e)  {
 }

 [CommandMethod("HelloObjectReactorAdd")]
 public void HelloObjectReactorAdd()
 {
 Document doc = Application.DocumentManager.MdiActiveDocument;  Editor ed =
doc.Editor;
 Database db = doc.Database;
 ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать,  если нужно
добавлять объекты в текущий лист, вместо ModelSpace

 if (m_LineId != ObjectId.Null)
 {
```



```
 ed.WriteMessage("\nРеакторы уже установлены. Снимите реакторы  командой
HelloDbReactorRemove");
 return;
 }

 PromptEntityOptions entOpt = new PromptEntityOptions("Выберите отрезок: ");
 PromptEntityResult entRes = ed.GetEntity(entOpt);  m_LineId =

entRes.ObjectId;

 using (Transaction trans =
db.TransactionManager.StartTransaction())
 {
 Entity ent = trans.GetObject(m_LineId, OpenMode.ForRead) as  Entity;

 if (!(ent.GetType() == typeof(Line)))
                     {
 m_LineId = ObjectId.Null;
```

```
 }

 db.SystemVariableWillChange +=
HelloDbReactor_SystemVariableWillChange;
 db.SystemVariableChanged +=
HelloDbReactor_SystemVariableChanged;

 db.ObjectErased += HelloDbReactor_ObjectErased;  db.ObjectModified +=

HelloDbReactor_ObjectModified;

 db.ObjectReappended += HelloDbReactor_ObjectReappended; db.ObjectUnappended
                     += HelloDbReactor_ObjectUnappended;

 db.ObjectAppended += HelloDbReactor_ObjectAppended;  db.ObjectAppended +=

HelloDbReactor_ObjectOpenedForModify;

 trans.Commit();
 }

 // TODO 5.2.1: Выбрать отрезок, запомнить его ObjectId в m_LineId

 // TODO 5.2.1: Подписаться на события объекта line при помощи  заготовок
HelloObjectReactor_*
 // Modified, Erased, OpenedForModify, Reappended, Unappended  }

 [CommandMethod("HelloObjectReactorRemove")]
 public void HelloObjectReactorRemove()
 {
 Document doc = Application.DocumentManager.MdiActiveDocument;  Editor ed =
doc.Editor;
 Database db = doc.Database;
 ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать,  если нужно
добавлять объекты в текущий лист, вместо ModelSpace

 if (m_LineId == ObjectId.Null)
 {
 return;
 }
```



```
 using (Transaction trans =
db.TransactionManager.StartTransaction())
 {
 Line line = trans.GetObject(m_LineId, OpenMode.ForRead) as  Line;

 if (line == null)
                     {
 ed.WriteMessage($"null\n");
                         return;
 }

 // TODO 5.2.2: Отписаться от событий объекта line // Modified, Erased,
                     OpenedForModify, Reappended, Unappended

 ed.WriteMessage($"Сейчас ко всему привяжемся! Не переживай!");

 db.SystemVariableWillChange -=
HelloDbReactor_SystemVariableWillChange;
 db.SystemVariableChanged -=
HelloDbReactor_SystemVariableChanged;

 db.ObjectErased -= HelloDbReactor_ObjectErased;  db.ObjectModified -=
```

```
HelloDbReactor_ObjectModified;

 db.ObjectReappended -= HelloDbReactor_ObjectReappended; db.ObjectUnappended -
                      = HelloDbReactor_ObjectUnappended;

 db.ObjectAppended -= HelloDbReactor_ObjectAppended;  db.ObjectAppended -=

HelloDbReactor_ObjectOpenedForModify;

 trans.Commit();
 }
 }

 void HelloEditorReactor_PointMonitor(object sender,
PointMonitorEventArgs e)
 {

 Document doc = Application.DocumentManager.MdiActiveDocument;  Database db =
doc.Database;


 using (Transaction tr = db.TransactionManager.StartTransaction())  {

 BlockTable bt = tr.GetObject(db.BlockTableId,  OpenMode.ForRead) as
BlockTable;
 BlockTableRecord btr =
tr.GetObject(bt[BlockTableRecord.ModelSpace], OpenMode.ForWrite) as
BlockTableRecord;
 Circle circle = new Circle(e.Context.RawPoint,  Vector3d.ZAxis, 10);


 btr.AppendEntity(circle);
                      tr.AddNewlyCreatedDBObject(circle, true);
```



```
e.Context.DrawContext.Geometry.Circle(e.Context.RawPoint, 10,  Vector3d.ZAxis);

 tr.Commit();
 }

 // TODO 5.5.1: Нарисовать окружность в точке курсора
(e.Context.RawPoint)
 // при помощи e.Context.DrawContext.Geometry.Circle()  }

 void HelloEditorReactor_LeavingQuiescentState(object sender,  EventArgs e)
 {
 }

 void HelloEditorReactor_EnteringQuiescentState(object sender,  EventArgs e)
 {
 }

 [CommandMethod("HelloEditorReactorAdd")]
 public void HelloEditorReactorAdd()
 {
 Document doc = Application.DocumentManager.MdiActiveDocument;  Editor ed =
doc.Editor;
 Database db = doc.Database;
 ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать,  если нужно
добавлять объекты в текущий лист, вместо ModelSpace

 ed.EnteringQuiescentState +=
HelloEditorReactor_EnteringQuiescentState;
 ed.LeavingQuiescentState +=
HelloEditorReactor_LeavingQuiescentState;
```

```
 ed.PointMonitor += HelloEditorReactor_PointMonitor;

 // TODO 5.5.1: Подписаться на события Editor
 // EnteringQuiescentState, LeavingQuiescentState, PointMonitor  }

 [CommandMethod("HelloEditorReactorRemove")]
 public void HelloEditorReactorRemove()
 {
 Document doc = Application.DocumentManager.MdiActiveDocument;  Editor ed =
doc.Editor;
 Database db = doc.Database;
 ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать,  если нужно
добавлять объекты в текущий лист, вместо ModelSpace

 ed.EnteringQuiescentState -=
HelloEditorReactor_EnteringQuiescentState;
 ed.LeavingQuiescentState -=
HelloEditorReactor_LeavingQuiescentState;
 ed.PointMonitor -= HelloEditorReactor_PointMonitor;

 // TODO 5.5.1: Отписаться от событий Editor
 // EnteringQuiescentState, LeavingQuiescentState, PointMonitor  }
```



```
 // DocumentCollection
 void HelloDocumentCollectionReactor_DocumentCreated(object sender,
DocumentCollectionEventArgs e)
 {
 HelloDocumentCollectionReactor_DocumentCreated(e.Document);  }

 void HelloDocumentCollectionReactor_DocumentCreated(Document doc)  {
 Database db = doc.Database;

 using (Transaction trans =
db.TransactionManager.StartTransaction())
 {
 BlockTable blockTable = trans.GetObject(db.BlockTableId,  OpenMode.ForRead) as
BlockTable;
 BlockTableRecord btr =
trans.GetObject(blockTable[BlockTableRecord.ModelSpace], OpenMode.ForRead) as
BlockTableRecord;

 // TODO 5.4.1: Подписаться на событие Modified блока  ModelSpace

 db.ObjectModified += HelloDbReactor_ObjectModified;  }
 }

 void HelloDocumentCollectionReactor_DocumentToBeDestroyed(object  sender,
DocumentCollectionEventArgs e)
 {
 Document doc = e.Document;

 Database db = doc.Database;

 using (Transaction trans =
db.TransactionManager.StartTransaction())
 {
 BlockTable blockTable = trans.GetObject(db.BlockTableId,  OpenMode.ForRead) as
BlockTable;
 BlockTableRecord btr =
trans.GetObject(blockTable[BlockTableRecord.ModelSpace], OpenMode.ForRead) as
BlockTableRecord;
 db.ObjectModified -= HelloDbReactor_ObjectModified; // TODO 5.4.1:
                     Отписаться от события Modified блока
ModelSpace
```

```
 }
 }
 // IExtensionApplication
 public void Initialize()
 {
 DocumentCollection docs = Application.DocumentManager;

 // TODO 5.4.1.A: Подписаться на события DocumentCollection:  DocumentCreated,
DocumentToBeDestroyed
 docs.DocumentCreated +=
HelloDocumentCollectionReactor_DocumentCreated;
 docs.DocumentToBeDestroyed +=
HelloDocumentCollectionReactor_DocumentToBeDestroyed;

 // Добавление реактора на изменение *Model_Space открытых на  момент загрузки
документов
```



```
 foreach (Document doc in docs)
 {
 HelloDocumentCollectionReactor_DocumentCreated(doc);  }
 }

 private void Docs_DocumentToBeDestroyed(object sender,
DocumentCollectionEventArgs e)
 {
 throw new NotImplementedException();
 }

 public void Terminate()
 {
 DocumentCollection docs = Application.DocumentManager;
docs.DocumentCreated -=
HelloDocumentCollectionReactor_DocumentCreated;
 docs.DocumentToBeDestroyed -=
HelloDocumentCollectionReactor_DocumentToBeDestroyed;

 foreach (Document doc in docs)
 {
 HelloDocumentCollectionReactor_DocumentCreated(doc);  }
 // TODO 5.4.1.A: Отписаться от событий DocumentCollection:  DocumentCreated,
DocumentToBeDestroyed
 }
 }

 }
}
//6 модуль WinForms

namespace HelloDotNET
{
 using System;
 using System.Collections.Generic;
 using System.Linq;
 using System.Text;

#if NCAD
 using Teigha.DatabaseServices;
 using Teigha.Runtime;
 using Teigha.Geometry;
 using HostMgd.Runtime;
 using HostMgd.ApplicationServices;
 using HostMgd.EditorInput;

 using Platform = HostMgd;
```

```
 using PlatformDb = Teigha;
#else
 using Autodesk.AutoCAD.ApplicationServices;
 using Autodesk.AutoCAD.DatabaseServices;
 using Autodesk.AutoCAD.EditorInput;
 using Autodesk.AutoCAD.Geometry;
 using Autodesk.AutoCAD.Runtime;
 using AcadApp = Autodesk.AutoCAD.ApplicationServices.Application;

 using Platform = Autodesk.AutoCAD;
 using PlatformDb = Autodesk.AutoCAD;
#endif
```



```
 public partial class Commands
 {
 [CommandMethod("HelloWinFormsModal")]
 public void HelloWinFormsModal()
 {
 Document doc = Application.DocumentManager.MdiActiveDocument;  Editor ed =
doc.Editor;
 Database db = doc.Database;
 ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать, если нужно
добавлять объекты в текущий лист, вместо ModelSpace

 // TODO 6.1.1.G: Вывести модальный диалог CopyDlgWinForms
Application.ShowModalDialog(new CopyDlgWinForms());  }

 [CommandMethod("HelloWinFormsModeless")]
 public void HelloWinFormsModeless()
 {
 Document doc = Application.DocumentManager.MdiActiveDocument;  Editor ed =
doc.Editor;
 Database db = doc.Database;
 ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать, если нужно
добавлять объекты в текущий лист, вместо ModelSpace

 // TODO 6.1.1.G: Вывести немодальный диалог CopyDlgWinForms
Application.ShowModelessDialog(new CopyDlgWinForms());  }

 static Platform.Windows.PaletteSet paletteSet;
 static Platform.Windows.Palette paletteWinForms;

 [CommandMethod("HelloWinFormsPalette")]
 public void HelloWinFormsPalette()
 {
 if (paletteSet == null)
 {
 // TODO 6.2.1.A: Создать набор палитр "HelloDotNET PaletteSet"  (набор палитр
общий для WinForms и WPF)
 // Указать GUID (new Guid("6AA64CDC-7CBA-49C5-8A14- 0B8775BAC5BC"))
 paletteSet = new Platform.Windows.PaletteSet("HelloDotNET  PaletteSet", new
Guid("6AA64CDC-7CBA-49C5-8A14-0B8775BAC5BC"));  // TODO: Задать свойства Size
и MinimumSize в 300x300  paletteSet.Size = new Size(800, 800);
 paletteSet.MinimumSize = new Size(300, 300);  }
 else
 {
 paletteSet.Visible = true;
 }
 if (paletteWinForms == null)
 {
 // TODO 6.2.1.B: Добавить объект PaletteWinFormsChild с именем "WinFormsPalette"
в набор палитр
 PaletteWinFormsChild paletteWinFormsChild= new
PaletteWinFormsChild();
 paletteSet.Add("WinFormsPalette", paletteWinFormsChild);
```

```
paletteSet.Visible = true;
 }
 }
```



```
 }
}
```

WPF

```
namespace HelloDotNET
{
 using System;
 using System.Collections.Generic;
 using System.Linq;
 using System.Text;

#if NCAD
 using Teigha.DatabaseServices;
 using Teigha.Runtime;
 using Teigha.Geometry;
 using HostMgd.Runtime;
 using HostMgd.ApplicationServices;
 using HostMgd.EditorInput;

 using Platform = HostMgd;
 using PlatformDb = Teigha;
#else
 using Autodesk.AutoCAD.ApplicationServices;
 using Autodesk.AutoCAD.DatabaseServices;
 using Autodesk.AutoCAD.EditorInput;
 using Autodesk.AutoCAD.Geometry;
 using Autodesk.AutoCAD.Runtime;
 using AcadApp = Autodesk.AutoCAD.ApplicationServices.Application;

 using Platform = Autodesk.AutoCAD;
 using PlatformDb = Autodesk.AutoCAD;
#endif

 public partial class Commands
 {
 [CommandMethod("HelloWPFModal")]
 public void HelloWPFModal()
 {
 Document doc = Application.DocumentManager.MdiActiveDocument;  Editor ed =
doc.Editor;
 Database db = doc.Database;
 ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать, если нужно
добавлять объекты в текущий лист, вместо ModelSpace

 // TODO: Вывести модальный диалог CopyDlgWpf
 Application.ShowModalWindow(new CopyDlgWpf());
 }

 [CommandMethod("HelloWPFModeless")]
 public void HelloWPFModeless()
 {
 Document doc = Application.DocumentManager.MdiActiveDocument;  Editor ed =
doc.Editor;
 Database db = doc.Database;
 ObjectId CurrentSpaceId = db.CurrentSpaceId; // Использовать, если нужно
добавлять объекты в текущий лист, вместо ModelSpace

 // TODO: Вывести немодальный диалог CopyDlgWpf
 Application.ShowModelessWindow(new CopyDlgWpf());  }
```

```
static Platform.Windows.Palette paletteWpf;

[CommandMethod("HelloWPFPalette")]
public void HelloWPFPalette()
{
if (paletteSet == null)
{
// TODO 6.2.1.A: Создать набор палитр "HelloDotNET PaletteSet"  (набор палитр общий для WinForms и WPF)
// Указать GUID (new Guid("6AA64CDC-7CBA-49C5-8A14- 0B8775BAC5BC"))
paletteSet = new Platform.Windows.PaletteSet("HelloDotNET  PaletteSet", new Guid("6AA64CDC-7CBA-49C5-8A14-0B8775BAC5BC"));  // TODO: Задать свойства Size и MinimumSize в 300x300  paletteSet.Size = new Size(800, 800);
paletteSet.sMinimumSize = new Size(300, 300);  }
else
{
paletteSet.Visible = true;
}
if (paletteWpf == null)
{
// TODO 6.2.1.B: Добавить объект PaletteWpfChild с именем  "WpfPalette" в набор палитр
paletteSet.AddVisual("WpfPalette", new PaletteWpfChild());

paletteSet.Visible = true;
}
}
}
```

Plot

```
namespace HelloDotNET
{
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

#if NCAD
using Teigha.DatabaseServices;
using Teigha.Runtime;
using Teigha.Geometry;
using HostMgd.Runtime;
using HostMgd.ApplicationServices;
using HostMgd.EditorInput;

using Platform = HostMgd;
using PlatformDb = Teigha;
using PlatformCOMApp = nanoCAD;

//using nanoCAD;
using OdaX;
#else
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;
```



```
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.Runtime;
using AcadApp = Autodesk.AutoCAD.ApplicationServices.Application;

using Platform = Autodesk.AutoCAD;
```

```
 using PlatformDb = Autodesk.AutoCAD;
 using PlatformCOMApp = AutoCAD;
#endif

 public partial class Commands
 {
 List<string> layouts = new List<string>();
 [CommandMethod("HelloPlot")]
 public void HelloPlot()
 {
 Document doc = Application.DocumentManager.MdiActiveDocument;

#if NCAD
 PlatformCOMApp.Document comDoc = doc.AcadDocument as
PlatformCOMApp.Document;
 PlatformCOMApp.Plot plot = comDoc.Plot;
#else
 PlatformCOMApp.AcadApplication comApp = Application.AcadApplication as
PlatformCOMApp.AcadApplication;
 PlatformCOMApp.AcadDocument comDoc = comApp.ActiveDocument as
PlatformCOMApp.AcadDocument;

 PlatformCOMApp.AcadPlot plot = comDoc.Plot;
#endif

 // TODO 6.3.1: A. Перебрать все листы при помощи foreach (AcadLayout  layout in
comDoc.Layouts)
 // Для каждого листа:
 foreach (AcadLayout layout in comDoc.Layouts)
 {
 layouts.Add(layout.Name);
 // B. Установить "Встроенный PDF-принтер" (поле ConfigName):
 layout.ConfigName = "Встроенный PDF-принтер";
 // C. Установить область печати (поле PlotType,  перечисление AcPlotType) с
проверкой типа листа:
 // если это пространство модели (ModelType=true),  установить acLimits, если нет
(ModelType=false) - acLayout
 if (layout.ModelType)
 layout.PlotType = AcPlotType.acLimits;
 else
 layout.PlotType = AcPlotType.acLayout;
 // D. Установить автоматический масштаб (поле StandardScale, перечисление
AcPlotScale, значение acScaleToFit)  layout.StandardScale =
AcPlotScale.acScaleToFit;  // E. Установить флаг стандартного масштаба (поле
UseStandardScale)
 layout.UseStandardScale = true;
 // F. Сделать лист активным
 comDoc.ActiveLayout = layout;

 // Специфичные настройки принтера "Встроенный PDF-принтер"

 // Запись специфичных настроек принтера в переменную
nanoCAD.InanoCADPlotCustomParams customPlotParams =
plot.CustomPlotSettings[layout];
```



```
 // Изменение специфичных настроек встроенного PDF-принтера
customPlotParams.RunPDFApp = false; // Не открывать PDF после  печати
 plot.CustomPlotSettings[layout] = customPlotParams; // Установить  изменённые
параметры

 // Распечатать лист при помощи PlotToFile(). В качестве параметра  в метод можно
передать:
 // 1. пустую строку - напечатанный документ будет сохранен в  папке, где лежит
dwg-файл и назван также
 // 2. имя файла - напечатанный файл будет назв ан, как указано в  параметре и
```

```
сохранен в одну папку с чертежом
 // 3. директорию - напечатанный файл бузет назван также, как  чертеж и сохранен
в указанную директорию
 // 4. директорию и имя файла :
plot.PlotToFile("C:\\PlotToFileExample\\myplot_<DN>_<LN>")
 plot.PlotToFile("C:\\Users\\1\\Documents\\LISTS");  // Внимание! Если
директория, указанная в параметре метода  PlotToFile() не существует, то будет
ошибка печати.

 }
 // TODO 6.3.3: Собрать названия листов в список
 // Установить перечень листов для печати вызовом метода  SetLayoutsToPlot()
 // Внимание! Plot.SetLayoutsToPlot() принимает на вход  массив листов.
 plot.SetLayoutsToPlot(layouts.ToArray());
 // Если листы собраны в список, нужно их преобразовать в  массив при помощи
ToArray().
 // Распечатать листы при помощи PlotToFile()  plot.PlotToFile("");
 }
 }
}
```

Подтверждаю, что отчет выполнен лично и соответствует требованиям практики
Балынин Е.Д.
Подпись: