

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления
Кафедра интеллектуальных информационных технологий

Инструкция
по выполнению лабораторной работы №6
по дисциплине «Проектирование баз знаний»

ПРИЛОЖЕНИЕ ДЛЯ РАБОТЫ С ОНТОЛОГИЯМИ

Студент гр. 121701

Р. В. Липский

Минск 2024

СОДЕРЖАНИЕ

| | | |
|-----|--|----|
| 1 | Требования к выполнению лабораторной работы | 2 |
| 1.1 | Требования к реализации лабораторной работы | 2 |
| 1.2 | Структура отчёта по лабораторной работе | 2 |
| 2 | Описание разрабатываемого приложения | 4 |
| 3 | Выполнение лабораторной работы | 5 |
| 3.1 | Создание проекта в IntelliJ IDEA | 5 |
| 3.2 | Подключение необходимых зависимостей | 7 |
| 3.3 | Реализация вкладок для интерфейса приложения | 9 |
| | Список использованных источников | 15 |

1 ТРЕБОВАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ

1.1 Требования к реализации лабораторной работы

Необходимо разработать графическое или web-приложение, использующее одно из хранилищ онтологий по выбранной предметной области.

Уточнение постановки задачи для лабораторной работы:

- Выбрать предметную область, для которой будет разрабатываться приложение, согласовать с преподавателем. Предметную область и требования к функциональным возможностям приложения можно взять в ЛР №2.

- Онтология должна содержать как иерархический набор понятий, так некоторое количество экземпляров понятий и связей между ними, достаточное для демонстрации работоспособности приложения

- Полученную онтологию погрузить в какое-либо из существующих RDF-хранилищ, развернутое локально или в облаке. Выбранное хранилище должно позволять осуществлять доступ к погруженной информации на чтение и запись. Примеры хранилищ: Virtuoso, Sesame, Jena. Полный список хранилищ с их сравнением: https://en.wikipedia.org/wiki/Comparison_of_triplestores

- Разработать графическое (десктопное) или web-приложение, позволяющее просматривать информацию из погруженной в хранилище онтологии, создавать в ней новые экземпляры и, при необходимости, классы, редактировать имеющиеся экземпляры и классы. Реализовать 3-4 сложных поисковых запроса по нескольким параметрам, ввод параметров и вывод результата должен осуществляться в форме, понятной конечному пользователю (должны использоваться графические компоненты управления пользовательским интерфейсом, вывод осуществляться в структурированном виде, например, в виде таблицы). Языки реализации и используемые дополнительные средства (библиотеки, фреймворки и т.д.) выбрать самостоятельно. Для доступа к хранилищу использовать общепринятые языки запросов и соответствующие библиотеки, реализующие возможности этих языков для выбранного языка программирования. Примеры таких языков: SPARQL (только чтение), GraphQL, Gremlin.

- Необходимо подготовить отчет с обоснованием выбора языков, библиотек и прочее, описание процесса работы, описание процесса установки выбранных средств, скриншоты проделанной работы.

1.2 Структура отчёта по лабораторной работе

а) Титульный лист

- б) Содержание
- в) 1-ый раздел – Обоснование выбора языка и средств реализации. Необходимо выбрать 3-4 языка и средства реализации сравнить. Сделать вывод о том, какой язык и средства реализации будут использованы.
- г) 2-ой раздел - Структура онтологии <название онтологии>. В рамках данного раздела необходимо:
- 1) Указать название онтологии, пояснить ее назначение
 - 2) Продемонстрировать, что в разработанной Онтологии существует иерархический набор понятий (классов), указать сколько каких понятий выделено
 - 3) Указать, сколько и каких выделено отношений
 - 4) Указать, сколько и каких описано экземпляров. С помощью каких отношений
 - 5) Продемонстрировать в виде скриншотов пункт 2 и 3. Каждый рисунок должен быть подписан.
 - 6) При необходимости разработки нескольких онтологий отразить для них п. 1-5
- д) 3-ий раздел – Описание разработанного приложения. В рамках данного раздела необходимо:
- Привести общую структуру приложения. Можно диаграмму классов.
 - Полученную онтологию погрузить в какое-либо из существующих RDF-хранилищ, развернутое локально или в облаке. Выбранное хранилище должно позволять осуществлять доступ к погруженной информации на чтение и запись. Необходимо обосновать выбор RDF-хранилища, которое будет использовано для погружения своей онтологии. Результат погружения описать словесно и прикрепить скрин.
 - Для доступа к хранилищу использовать общепринятые языки запросов и соответствующие библиотеки, реализующие возможности этих языков для выбранного языка программирования. Примеры таких языков: SPARQL (только чтение), GraphQL, Gremlin.
 - Описать и обосновать какие языки будут использованы для работы с хранилищем.
 - Привести алгоритм установки и настройки разработанного вами приложения. Как запускать, как подключать библиотеки. Как загружать и работать с онтологией.
 - Привести примеры запросов к онтологии.
 - Привести в целом примеры (скрины) работы с системой. На каждый рисунок должно быть описание в тексте.
- е) Вывод – кратко, что было сделано, результаты
- ж) Список используемых источников – минимум 5

2 ОПИСАНИЕ РАЗРАБАТЫВАЕМОГО ПРИЛОЖЕНИЯ

В рамках данной инструкции будет приведен пример разработки простого редактора OWL-онтологий, который позволит нам

- читать существующие онтологии;
- сохранять отредактированную онтологию;
- создавать, удалять и редактировать классы;
- создавать удалять и редактировать объекты классов;
- создавать, удалять и редактировать примитивные и объектные отношения;
- выполнять SPARQL-запросы.

Для реализации лабораторной работы будут использоваться следующие средства:

- IDE – IntelliJ IDEA
- Язык программирования – Kotlin
- Фреймворк для создания пользовательского интерфейса – Jetpack Compose
- RDF-хранилище – Apache Jena.

Для того, чтобы следовать данной инструкции вам понадобится самостоятельно установить IntelliJ IDEA, для чего вы можете воспользоваться инструкцией, расположенной на <https://javarush.com/quests/lectures/questsyntaxpro.level20.lecture01>.

3 ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ

3.1 Создание проекта в IntelliJ IDEA

Для начала выполнения лабораторной работы, после установки IntelliJ IDEA, запустите данную IDE – после запуска вас встретит приветственный экран, изображенный на рисунке 3.1.

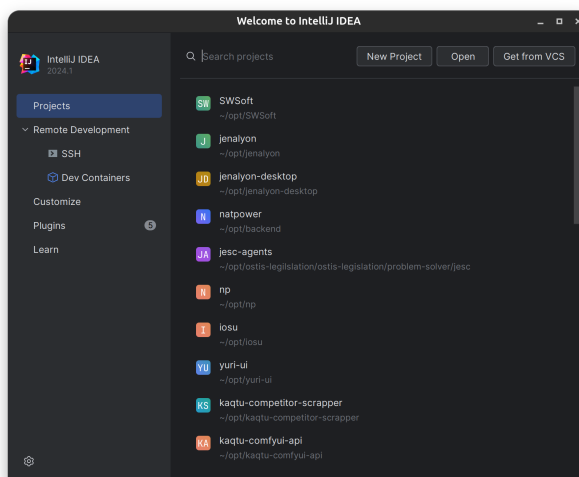


Рисунок 3.1 – Приветственный экран IntelliJ IDEA

Чтобы начать разработку, найдите в правом верхнем углу кнопку «New Project» и нажмите на неё. Данное действие откроет вам экран мастера создания проектов, изображенный на рисунке 3.2.

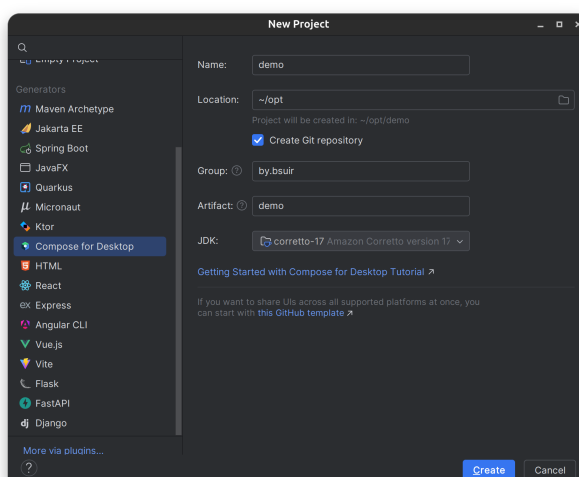


Рисунок 3.2 – Экран мастера создания проектов IntelliJ IDEA

В левом боковом меню мастера создания проектов представлены шаблоны для разработки проектов – по сути, это заготовленный набор файлов для быстрого начала разработки с использованием определенных

библиотек и фреймворков. Как уже упоминалось выше, для разработки пользовательского интерфейса мы будем использовать Jetpack Compose, потому вам необходимо выбрать шаблон «Compose for Desktop» из вкладки «Generators» так, как это изображено на рисунке 3.2.

В поле «name» вы можете ввести имя своего проекта. Поля «group», «artifact» и «JDK» оставьте без изменений, в случае, если только вы не профессиональный разработчик на Kotlin, который знает, что они означают. После заполнения данных полей нажмите кнопку «Create», чтобы создать шаблон вашего проекта.

После создания вас встретит основное окно IntelliJ IDEA, изображенное на рисунке 3.3, где мы и проведем большую часть времени разработки нашего приложения. Давайте отдельно остановимся на данном окне и рассмотрим, из чего оно состоит.

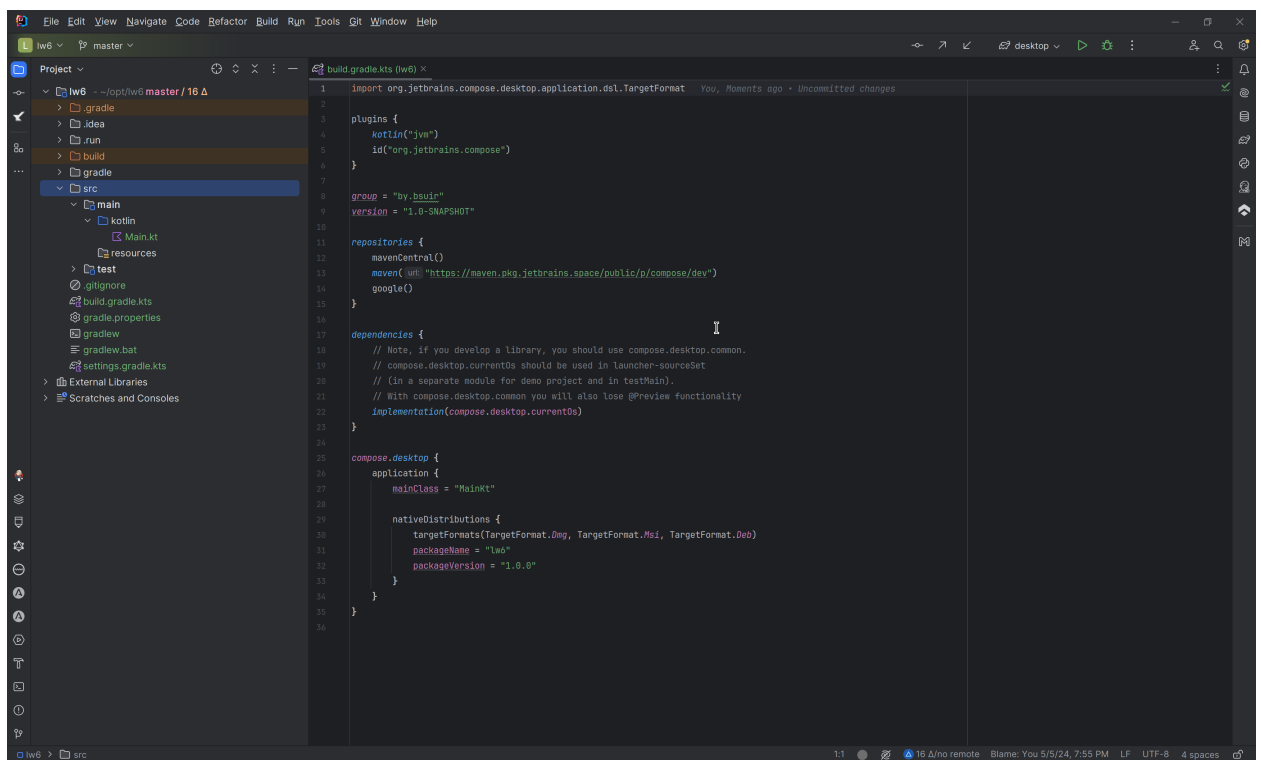


Рисунок 3.3 – Основное окно IntelliJ IDEA

Слева вы можете видеть дерево директорий, совсем как в файловом проводнике вашего компьютера. Работает оно точно также – вы видите папки и файлы, содержащиеся в вашем проекте. По нажатию на любой файл поддерживаемого формата, он откроется в правой части вашего приложения, где вы сможете его отредактировать! Именно так и разрабатываются приложения.

Справа сверху вы можете найти зеленую треугольную кнопку, которая запускает ваше приложение. Давайте нажмём её и посмотрим, что произойдёт.

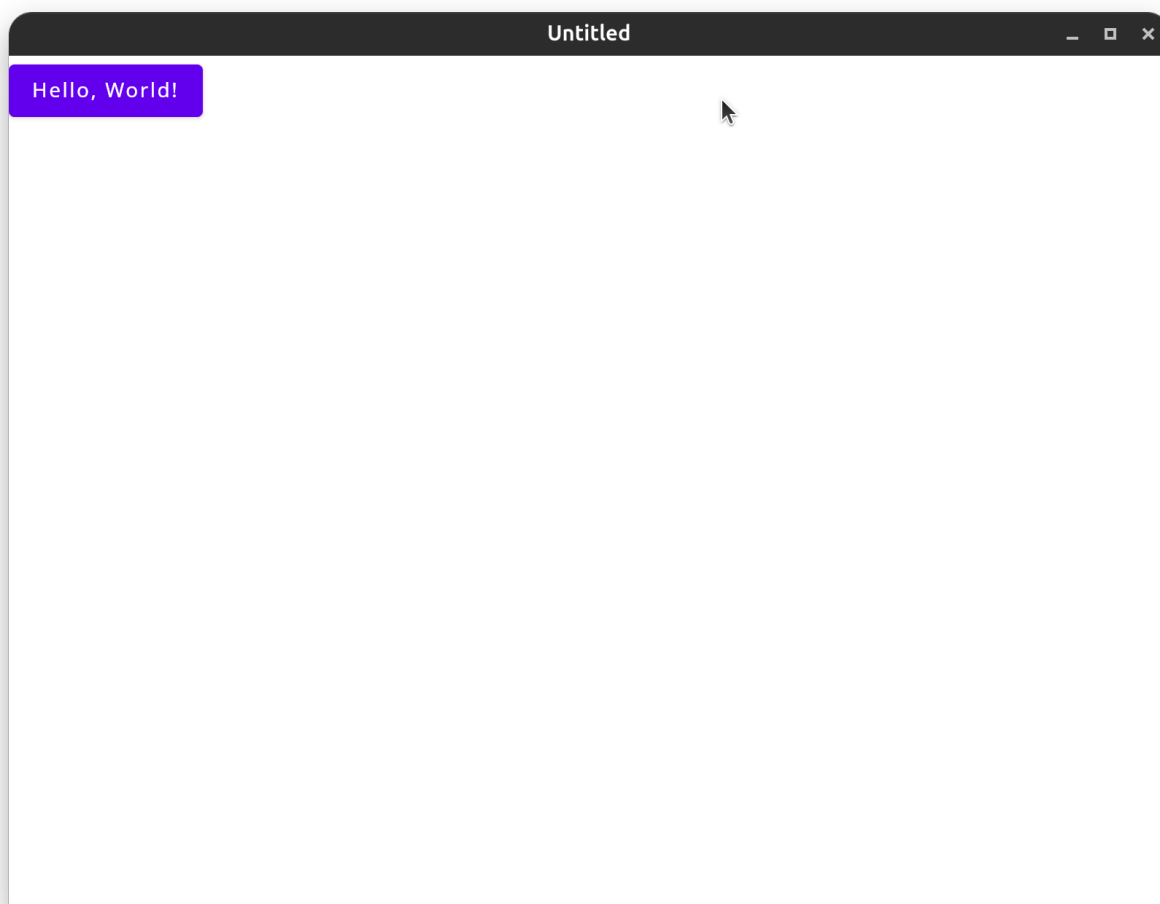


Рисунок 3.4 – Стандартное приложение

Поздравляем! Вы запустили заготовку приложения, которое мы будем разрабатывать в данной лабораторной работе и завершили этап создания проекта.

Если вы честно прошли данный этап инструкции, попробуйте ответить на следующие вопросы и проверить себя:

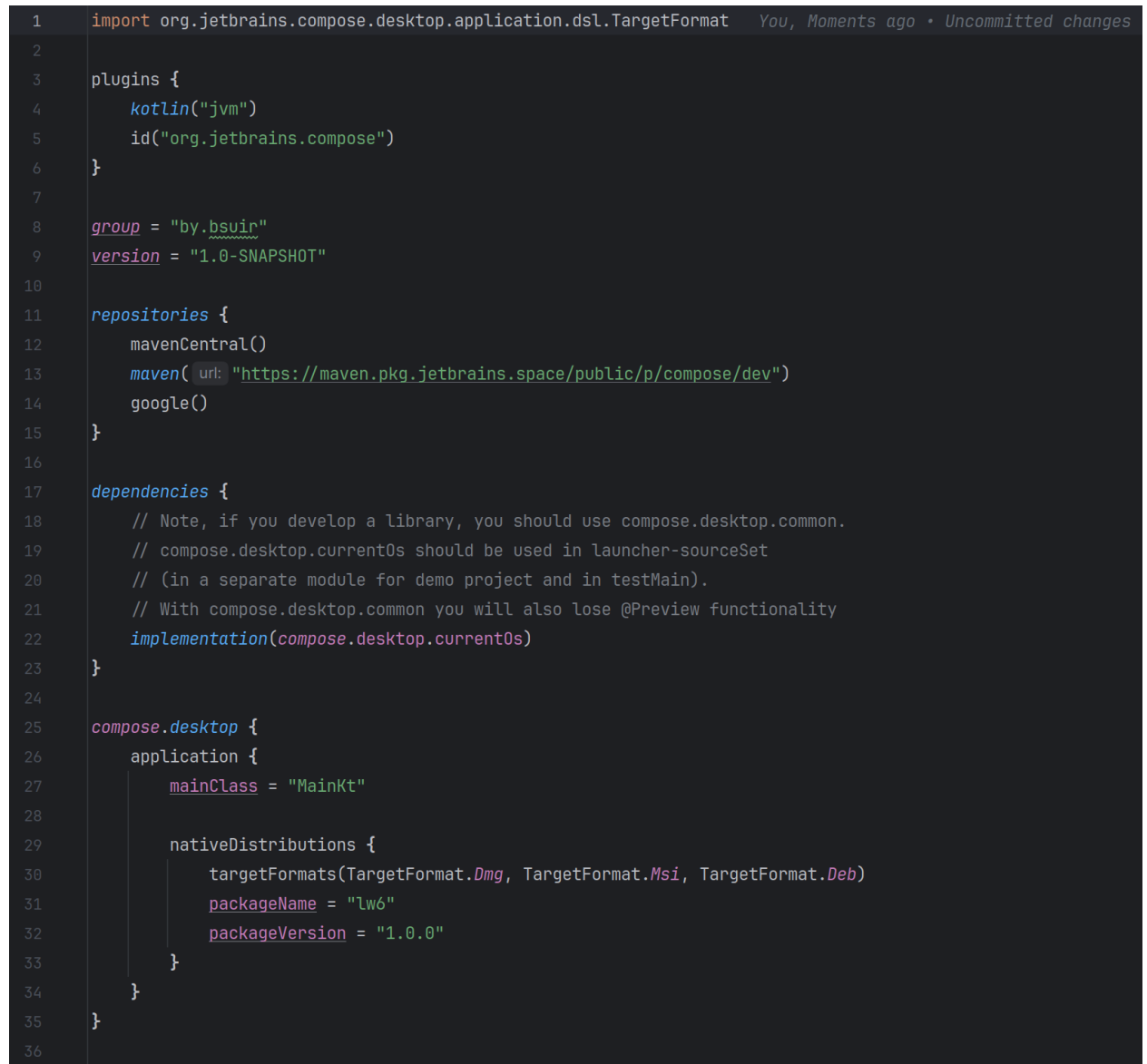
- а) Зачем был выбран «Compose for Desktop» на этапе создания проекта?
- б) Что отображается в левом боковом меню основного окна IntelliJ IDEA?
- в) Что делает зеленая треугольная кнопка в правом верхнем углу IntelliJ IDEA?

3.2 Подключение необходимых зависимостей

Для того, чтобы в нашем приложении мы могли работать с онтологиями, нам необходимо воспользоваться помощью умных людей, которые разработали для нас специальную библиотеку для работы с онтологиями.

Она называется **Apache Jena** и предоставит для нас готовый функционал для создания и редактирования онтологий.

В Kotlin добавлять библиотеки очень просто! Найдите в корне проекта файл, под названием «build.gradle.kts» и откройте его. Вы должны увидеть текст, похожий на то, что изображено на рисунке 3.5.



```
1 import org.jetbrains.compose.desktop.application.dsl.TargetFormat
2
3 plugins {
4     kotlin("jvm")
5     id("org.jetbrains.compose")
6 }
7
8 group = "by.bsuir"
9 version = "1.0-SNAPSHOT"
10
11 repositories {
12     mavenCentral()
13     maven(url = "https://maven.pkg.jetbrains.space/public/p/compose/dev")
14     google()
15 }
16
17 dependencies {
18     // Note, if you develop a library, you should use compose.desktop.common.
19     // compose.desktop.currentOs should be used in launcher-sourceSet
20     // (in a separate module for demo project and in testMain).
21     // With compose.desktop.common you will also lose @Preview functionality
22     implementation(compose.desktop.currentOs)
23 }
24
25 compose.desktop {
26     application {
27         mainClass = "MainKt"
28
29         nativeDistributions {
30             targetFormats(TargetFormat.Dmg, TargetFormat.Msi, TargetFormat.Deb)
31             packageName = "lw6"
32             packageVersion = "1.0.0"
33         }
34     }
35 }
36
```

Рисунок 3.5 – Содержимое файла build.gradle.kts

Данный файл представляет собой инструкцию по сборке нашего приложения. Он состоит из нескольких блоков, но нас интересует только один из них – «dependencies». Здесь мы поместим список библиотек, которые понадобятся нам для разработки нашего приложения.

Важно! Ни в коем случае не удаляйте и не редактируйте здесь ничего, если вы не знаете, зачем это нужно.

Для нашей работы мы добавим лишь одну строчку в блок «dependencies»:

```
implementation("org.apache.jena:apache-jena-libs:5.0.0")
```

Давайте разберем данную строку:

- `implementation` – означает, что данная зависимость понадобится нам при запуске приложения;
- `org.apache.jena` – идентификатор разработчика данной библиотеки;
- `apache-jena-libs` – название библиотеки;
- `5.0.0` – её версия.

Поняли? Отлично, теперь мы готовы приступить к разработке приложения.

3.3 Реализация вкладок для интерфейса приложения

Приступим к разработке пользовательского интерфейса приложения. Поскольку в приложении будут реализованы несколько групп функциональности (для классов, отношений, объектов, работы с онтологиями) мы создадим для каждой из них отдельные вкладки.

Для этого перейдите в файл «Main.kt» расположенный в директории «src/main/kotlin» и найдите в нём функцию «App». Её содержимое нам не понадобится, поэтому свободно удаляйте его.

Давайте напишем код для отображения меню выбора вкладок. Для этого разберемся в том, что из себя представляет написание интерфейса при помощи Jetpack Compose: каждый элемент интерфейса представляет из себя функцию в исходной коде, помеченную аннотацией `@Composable`.

Каждая функция, представляющая компонент интерфейса содержит в себе специальные команды для создания элементов. Если вы знакомы с версткой на HTML, то узнаете знакомый подход. Перейдём к конкретному примеру: скопируйте и вставьте в исходный код следующую функцию:

Листинг 1 – Компонент меню выбора вкладки

```
@Composable
fun Header() {
    TabRow(selectedTabIndex = selectedTab) {
        Tab(
            selected = selectedTab == 0,
            onClick = { selectedTab = 0 },
            modifier = Modifier.padding(8.dp)
        ) {
            Text("Classes")
        }

        Tab(
            selected = selectedTab == 1,
```

```

        onClick = { selectedTab = 1 },
        modifier = Modifier.padding(8.dp)
    ) {
        Text("Instances")
    }

    Tab(
        selected = selectedTab == 2,
        onClick = { selectedTab = 2 },
        modifier = Modifier.padding(8.dp)
    ) {
        Text("Primitive Props")
    }

    Tab(
        selected = selectedTab == 3,
        onClick = { selectedTab = 3 },
        modifier = Modifier.padding(8.dp)
    ) {
        Text("Object Props")
    }

    Tab(
        selected = selectedTab == 4,
        onClick = { selectedTab = 4 },
        modifier = Modifier.padding(8.dp)
    ) {
        Text("Ontology")
    }
}

```

Давайте разберем, что здесь написано:

- TabRow – компонент, предоставляющий интерфейс для выбора вкладки.

- Tab – отдельная вкладка, которую мы хотим создать

- Text – компонент, который помещает в заголовок вкладки текст

Для того, чтобы увидеть, что мы натворили, поместите внутрь основной функции App следующий код:

Листинг 2 – Код основной функции App для отображения меню выбора вкладок

```
var selectedTab by mutableStateOf(0)
```

```

@Composable
@Preview
fun App() {
    MaterialTheme {
        Column( Modifier . fillMaxSize () ) {
            Header ()
        }
    }
}

```

Таким образом после запуска приложения вы должны увидеть окно, изображенное на рисунке 3.6.

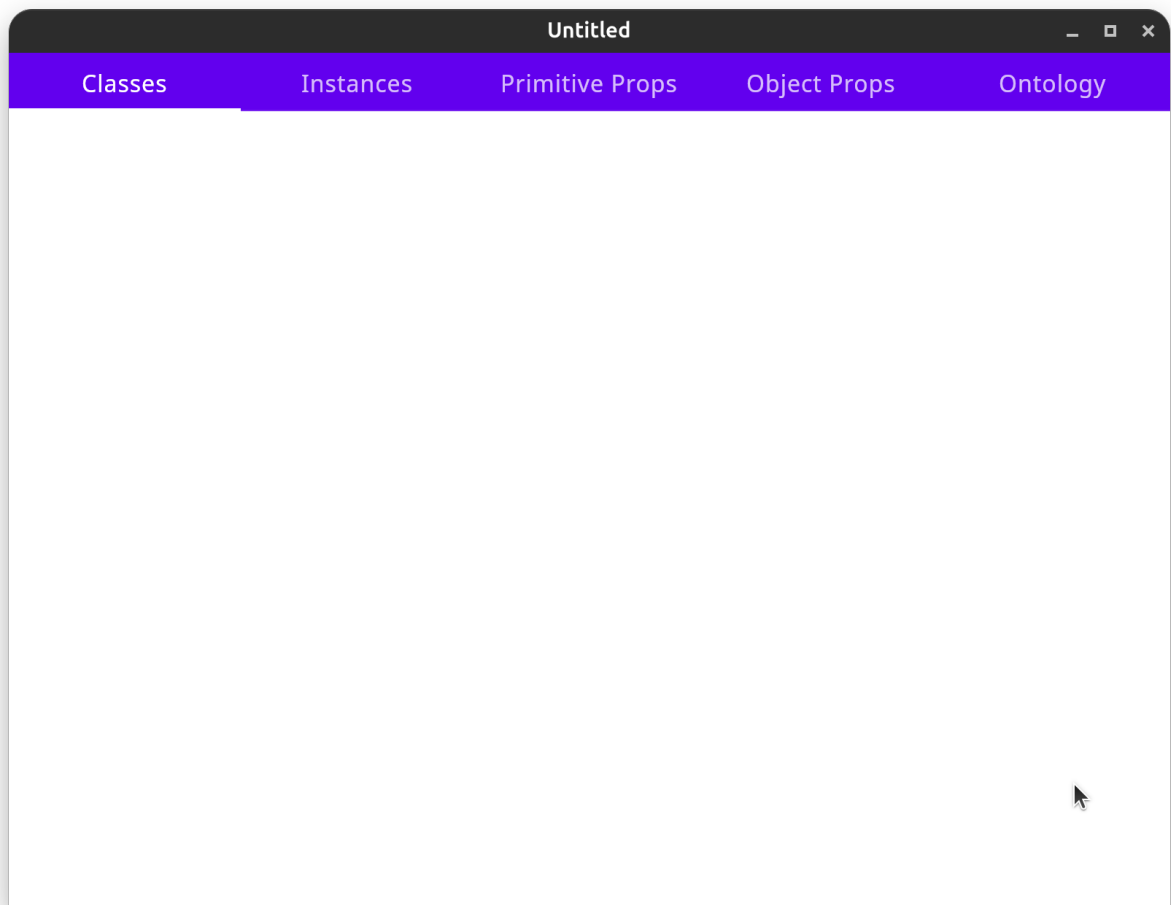


Рисунок 3.6 – Заготовка верхнего меню приложения

Начнём реализовывать вкладки нашего приложения. Начнём с простейшего - попытаемся просмотреть список всех классов, которые есть в нашей онтологии. Для этого создадим новый компонент, который назовем `ClassList`. Добавьте в корень исходного кода следующие строки:

```

var ontology: OntModel = ModelFactory
    .createOntologyModel(OntModelSpec.OWL_DL_MEM)

var classList by mutableStateOf(mutableListOf(
    *ontology.listClasses().toList().toTypedArray()
))

fun updateClassList() {
    classList = mutableListOf(
        *ontology.listClasses()
            .toList()
            .toTypedArray()
    )
}

```

Ontology – переменная, которая будет содержать в себе модель онтологии, classList – переменная, которая будет содержать список классов нашего приложения, updateClassList() - функция, которая будет обновлять содержимое classList, основываясь на тех классах, которые на данный момент загружены в модель онтологии.

Теперь нужно создать компонент, который будет отображать наш список классов:

```

@Composable
fun ClassesTab() {
    var showDialog by mutableStateOf(false)
    var className by mutableStateOf("")

    Box( Modifier.fillMaxWidth().padding(8.dp)) {
        LazyColumn {
            item {
                Button(
                    onClick = {
                        showDialog = true;
                        println("lol")
                    },
                    modifier = Modifier.fillMaxWidth()
                ) {
                    Text("Create new class")
                }

                if (showDialog) {

```

```

Dialog(onDismissRequest = {
    showDialog = false
}) {
    Card(
        modifier = Modifier
            .fillMaxWidth()
            .height(200.dp)
            .padding(8.dp),
        shape = RoundedCornerShape(16.dp),
    ) {
        Column(
            modifier = Modifier
                .padding(8.dp)
                .fillMaxSize()
        ) {
            Text(
                text = "Enter class name",
                modifier = Modifier
                    .fillMaxWidth()
            )
            Spacer(Modifier.height(8.dp))
            TextField(
                value = className,
                onValueChange = {
                    className = it
                },
                modifier = Modifier
                    .fillMaxWidth()
            )
            Button(
                modifier = Modifier
                    .fillMaxWidth(),
                onClick = {
                    showDialog = false
                    ontology
                        .createClass("http://je
updateClassList()
                }
            ) { Text("Create") }
        }
    }
}

```

```

    }
}

item {
    Row(
        Modifier
            .fillMaxWidth()
            .background(Color.LightGray)
            .padding(8.dp)
    ) {
        Text("Class")
        Text("")
    }
}

itemsIndexed(classList) { _, clazz ->
    Row {
        Column(
            modifier = Modifier
                .fillMaxWidth(0.9F)
        ) {
            Text(
                clazz.uri,
                modifier = Modifier
                    .padding(8.dp)
            )
        }

        Column {
            IconButton(
                onClick = {
                    clazz.remove()
                    updateClassList()
                }
            ) {
                Icon(Icons.Filled.Delete,
                    contentDescription = "Delete"
                )
            }
        }
    }
}
}

```

```

    }
  }
}

```

Теперь обновим код основного элемента интерфейса, чтобы при выборе первой вкладки, данная таблица отображалась пользователю:

```

MaterialTheme {

    Column( Modifier . fillMaxSize ( ) ) {

        Header ( )

        Spacer ( Modifier . height ( 16 . dp ) )

        when ( selectedTab ) {
            0 -> ClassesTab ( )
        }

    }

}

```

С остальными вкладками делаем также. И все.

