

Лабораторная работа 2

МОДЕЛИРОВАНИЕ ПРОГРАММНЫХ СИСТЕМ СРЕДСТВАМИ RATIONAL ROSE

Цель работы – приобрести навыки моделирования баз данных на примере построения диаграмм классов и деятельности, а также генерации программного кода на примере проектирования простого графического редактора.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Система Rational Rose – признанный лидер среди средств визуального моделирования, используя ее можно интерактивно разрабатывать архитектуру создаваемого приложения, генерировать его исходные тексты и параллельно работать над документированием разрабатываемой системы.

Преимущества применения Rational Rose:

- сокращение цикла разработки приложения;
- увеличение продуктивности работы программистов;
- улучшение потребительских качеств создаваемых программ за счет ориентации на пользователей и бизнес;
- способность вести большие проекты и группы проектов;
- возможность повторного использования уже созданного ПО за счет упора на разбор их архитектуры и компонентов.

Диаграммы, использующиеся в работе

Class diagram (диаграммы классов). Этот вид диаграмм позволяет создавать логическое представление системы, на основе которого создается исходный код описанных классов. Диаграмма классов служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования и содержит детальную информацию об архитектуре программной системы.

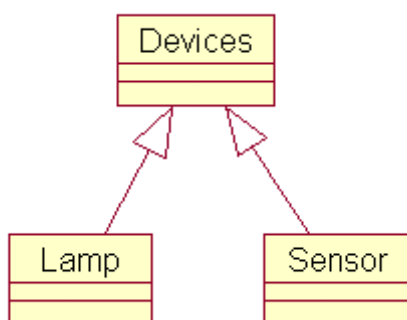


Рисунок 5.1 – Пример диаграммы классов

Activity diagram (диаграммы деятельности или активности). Каждый объект системы, обладающий определенным поведением, может находиться в определенных состояниях, переходить из состояния в состояние, совершая определенные действия в процессе реализации сценария поведения объекта. Поведение большинства объектов реальных систем можно представить с точки зрения теории конечных автоматов, то есть поведение объекта отражается в его состояниях. Для графического отображения состояния объекта используется два вида диаграмм: **Statechart diagram** (диаграмма состояний) и **Activity diagram** (диаграмма активности).

Диаграмма состояний предназначена для отображения состояний объектов системы, имеющих сложную модель поведения.

Диаграмма активности представляет собой дальнейшее развитие диаграммы состояний. Этот тип диаграмм позволяет показать не только последовательность процессов, но и ветвление и даже синхронизацию процессов, позволяет проектировать алгоритмы поведения объектов любой сложности, в том числе может использоваться для составления блок-схем.

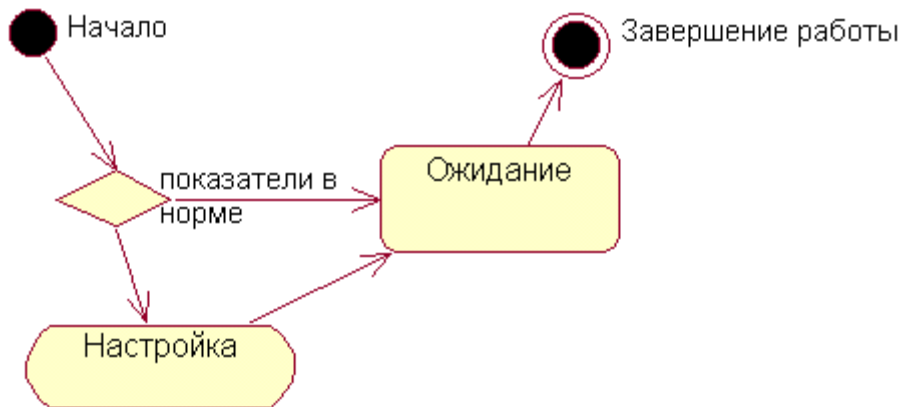


Рисунок 5.2 – Пример диаграммы деятельности

Component diagram (диаграммы компонентов). Этот тип диаграмм предназначен для распределения классов и объектов по компонентам при физическом проектировании системы. Часто данный тип диаграмм называют диаграммами модулей. Диаграмма компонентов служит частью физического представления модели и является необходимой для генерации программного кода.

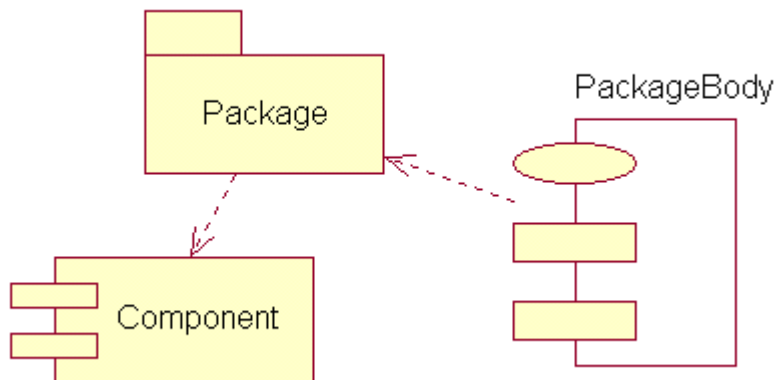


Рисунок 5.3 – Пример диаграммы компонентов

Отношения на диаграммах

Между компонентами диаграмм могут существовать различные отношения, которые описывают их взаимодействие. В языке UML имеется несколько стандартных видов отношений.

Ассоциация – это структурное двунаправленное отношение, описывающее совокупность взаимоотношений между объектами. Ассоциация может иметь имя и кратность. Кратность (multiplicity) ассоциации указывается рядом с обозначением компонента диаграммы, который является участником данной ассоциации. Кратность характеризует общее количество конкретных экземпляров данного компонента, которые могут выступать в качестве элементов данной ассоциации. Наиболее распространенными являются следующие формы записи кратности отношения ассоциации:

1. Целое неотрицательное число (включая цифру 0). Предназначено для указания кратности, которая является строго фиксированной для элемента соответствующей ассоциации.

2. Два целых неотрицательных числа, разделенные двумя точками и записанные в виде: "первое число .. второе число", например, 1..5. Очевидно, что первое число должно быть строго меньше второго числа в арифметическом смысле, при этом первое число может быть равно 0.

3. Два символа, разделенные двумя точками. При этом первый из них является целым неотрицательным числом или 0, а второй – символом "*" или "n". Здесь символ "*" или "n" обозначает произвольное конечное целое неотрицательное число, значение которого неизвестно на момент задания соответствующего отношения ассоциации.

4. Единственный символ "*" или "n". Является сокращением записи интервала формы записи 3.

Если кратность отношения ассоциации не указана, то по умолчанию принимается ее значение, равное 1.

Частным случаем ассоциации является отношение типа "часть/целое". Отношение такого типа называется **агрегированием**. Агрегирование изображается в виде ассоциации с незакрашенным ромбом со стороны целого.

Обобщение – это однонаправленное отношение, называемое "потомок/прародитель", в котором объект "потомок" может быть подставлен вместо объекта прародителя (родителя или предка). Потомок наследует структуру и поведение своего родителя. Графически обозначается сплошной линией со стрелкой в форме незакрашенного треугольника. Стрелка всегда указывает на родителя.

РАБОЧИЕ ЗАДАНИЯ

Задание 1. Построить диаграмму, изображающую логическую схему базы данных, на примере базы данных автоматизированной информационной системы регистрации учебных курсов. Порядок выполнения:

1.1. В новом окне самостоятельно создать диаграмму классов, которая состоит из следующих классов:

- Вуз;
- Факультет;
- Студент;
- Курс;
- Преподаватель.

1.2. Для созданных классов установить атрибуты как показано на рис. 5.4.

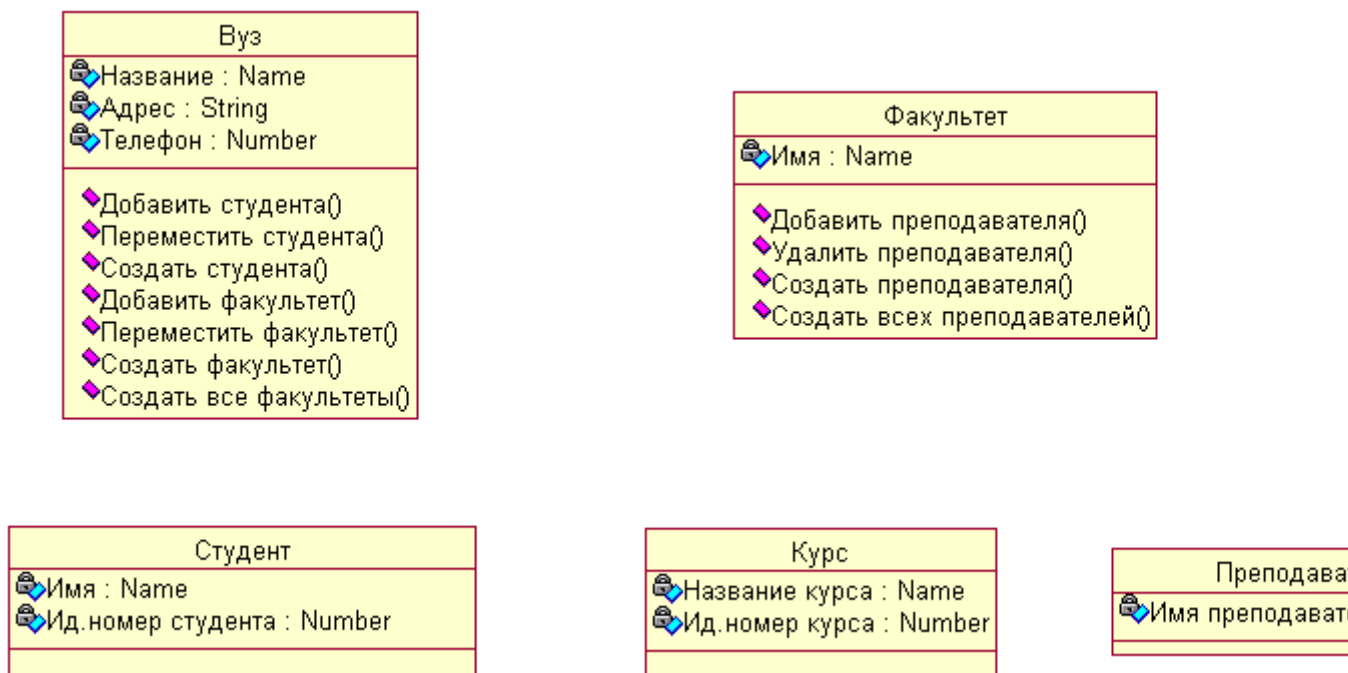


Рисунок 5.4 – Диаграмма классов с атрибутами

1.3. Установить взаимосвязи между классами в соответствии с рис. 5.5. Если кнопка агрегации отсутствует на специальной панели инструментов, ее следует добавить с помощью операции контекстного меню **Customize**.

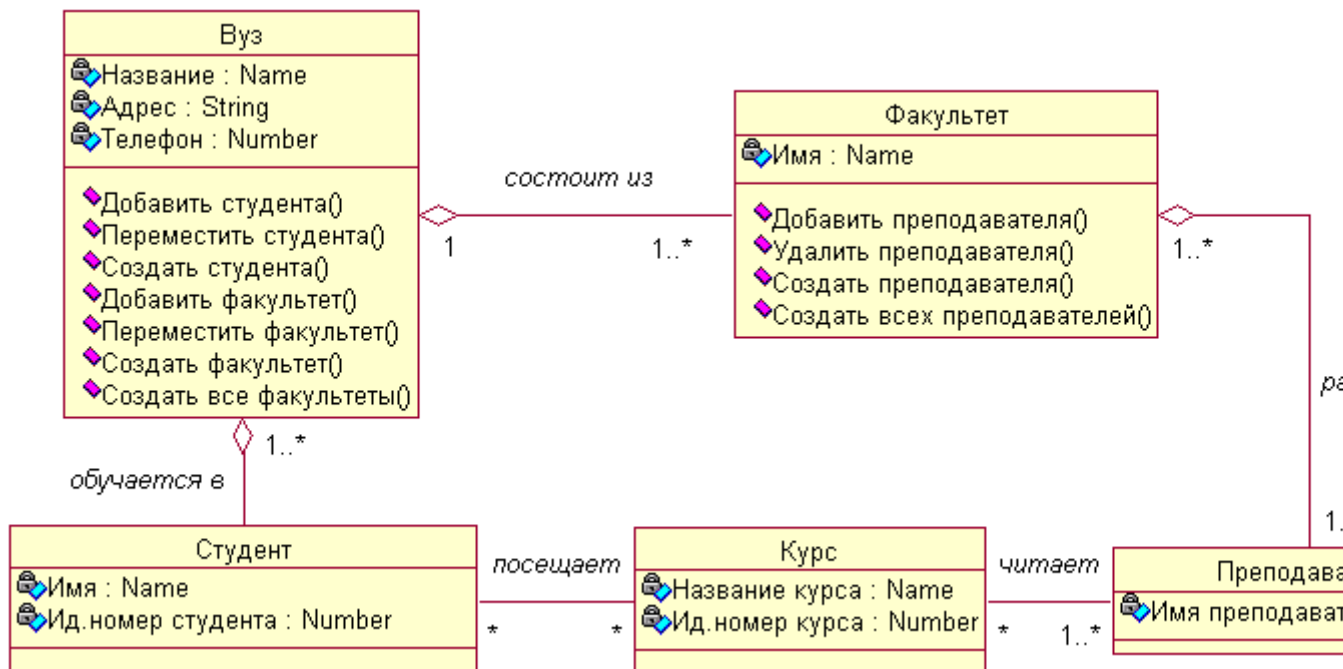


Рисунок 5.5 – Диаграмма классов, моделирующая объекты системы регистрации курсов и отношения между ними

Задание 2. Построить диаграмму деятельности, изображающую процесс "Открытие регистрации" в автоматизированной информационной системе регистрации учебных курсов. Порядок выполнения:

2.1. Открыть рабочее окно для построения диаграммы деятельности **Activity Diagram** с помощью контекстного меню представления **Use Case View** в браузере проекта.

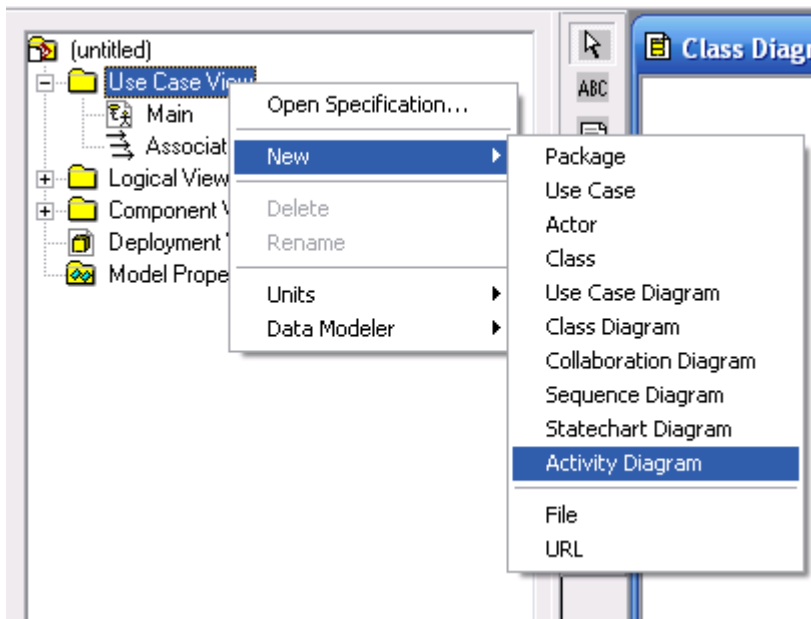


Рисунок 5.6 – Вид контекстного меню создания диаграммы деятельности

В результате появится новое окно с чистым рабочим листом диаграммы деятельности, а на специальной панели инструментов – кнопки, необходимые для разработки диаграммы деятельности. Назначение отдельных кнопок панели можно узнать из всплывающих подсказок.

2.2. Создать деятельности в окне браузера проекта в соответствии с рисунком 5.7.

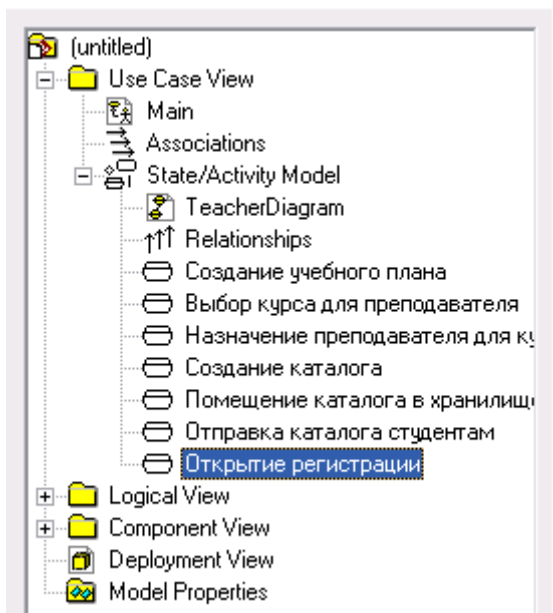


Рисунок 5.7 – Окно браузера проекта

2.3. Созданные деятельности перетащить в окно диаграммы деятельности и расположить как на рисунке 5.8.

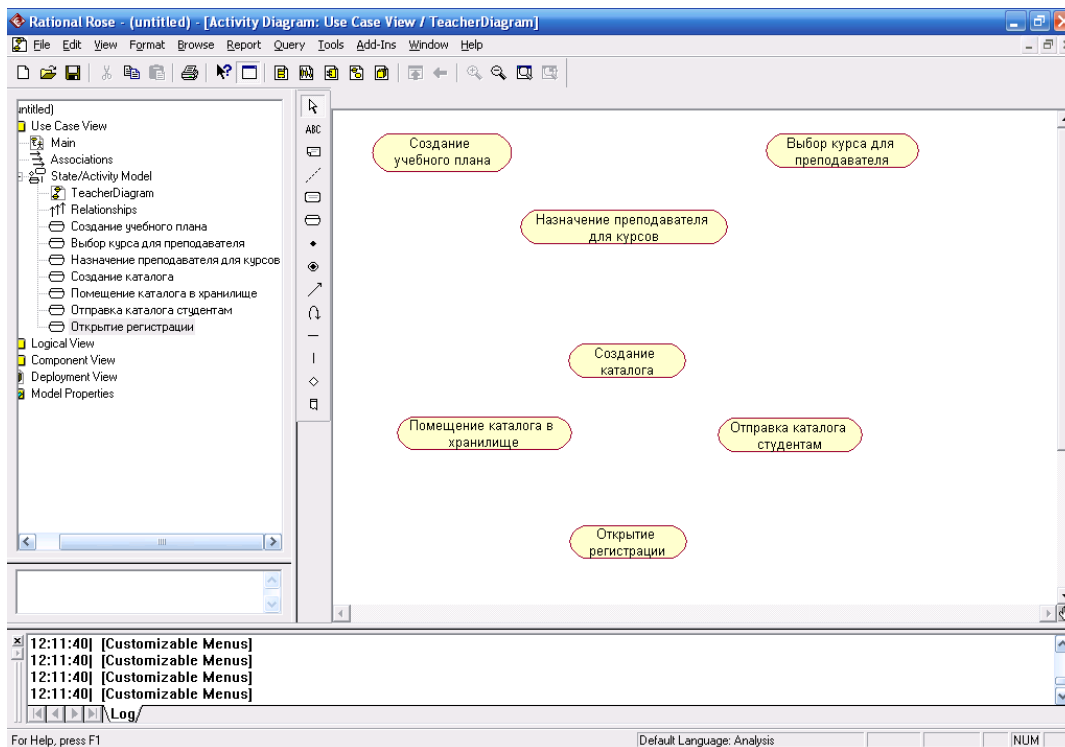



Рисунок 5.8 – Окно диаграммы деятельности

2.4. С помощью специальной панели инструментов добавить на диаграмму элемент принятия решения (ветвления) для альтернативных переходов  и расположить его как на рис. 5.9.

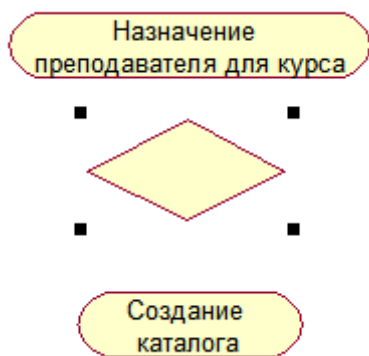


Рисунок 5.9 – Фрагмент окна диаграммы деятельности

2.5. С помощью команды **Open Specification ...** контекстного меню элемента ветвления задать вопрос условия перехода: "Все ли преподаватели назначены?" (рис. 5.10).

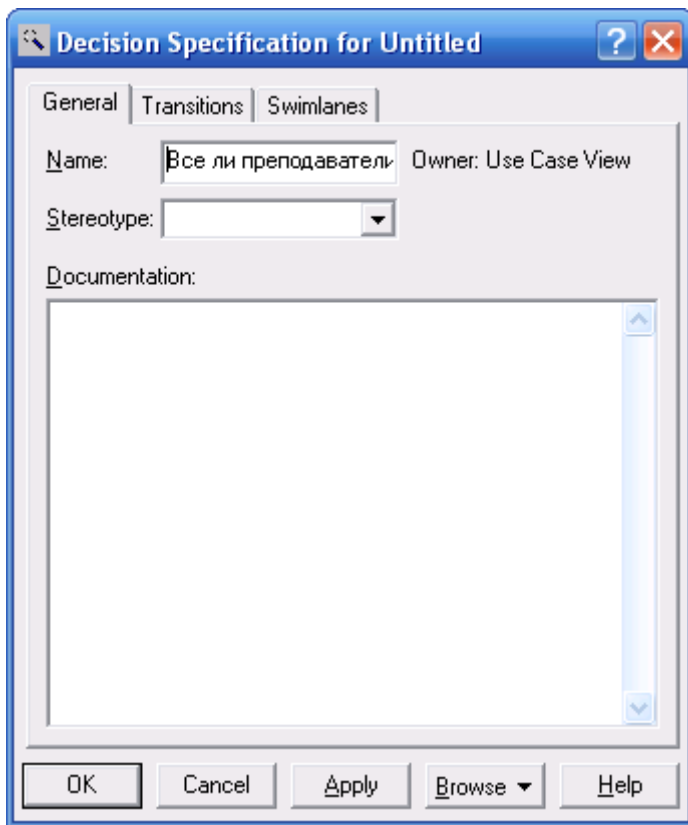


Рисунок 5.10 – Диалоговое окно свойств объекта ветвления диаграммы



2.6. С помощью специальной панели инструментов добавить на диаграмму синхронизационную черту  для отображения условия, соответствующего логическому оператору "и", а также переходы между деятельностями  как на рис. 5.11.



Рисунок 5.11 – Диаграмма деятельности, которая моделирует действия, выполняемые в процессе создания системы регистрации учебных курсов

Задание 3. Построить диаграмму классов, изображающую простой графический редактор. В графическом редакторе использовать следующие основные классы:

1. Figure (фигура);
2. Figure Element (элемент фигуры);
3. Point (точка);
4. Line (линия).

Порядок выполнения:

3.1. Создать и расположить все описанные выше классы в соответствии с рис. 5.12.

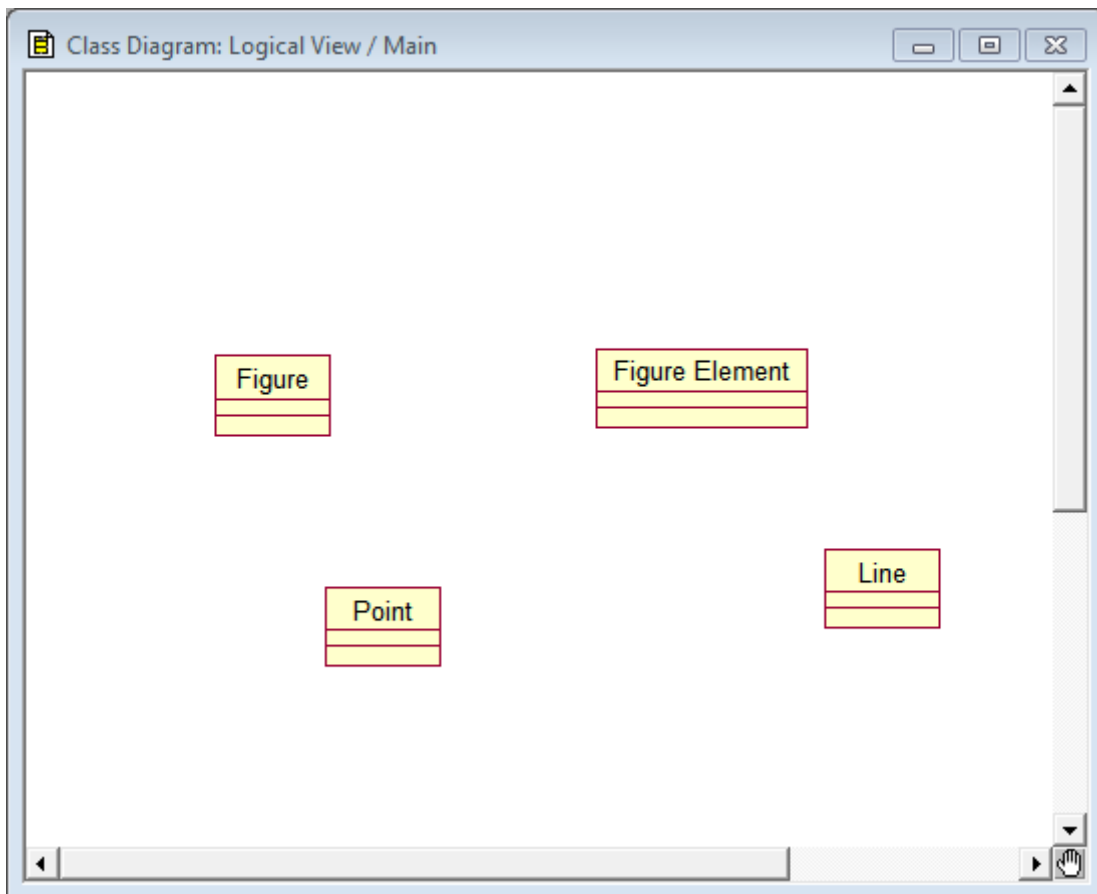



Рисунок 5.12 – Окно диаграммы классов графического редактора

3.2. Созданной диаграмме дать имя **Display**. Для этого воспользоваться инструментом **TextBox**  на специальной панели инструментов.

3.3. Изменить стиль форматирования надписи с помощью команд **Format...** → **Font...** из контекстного меню надписи. Установить шрифт полужирный, подчеркнутый, размер – 12 пт.

3.4. Для класса **Figure** добавить операции как на рис. 5.13.

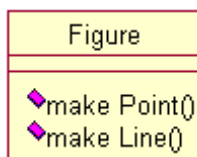


Рисунок 5.13 – Операции класса **Figure**

3.5. Для класса **Figure Element** добавить операцию с аргументами **move By(int , int)** . Операция **move By** перемещает объект в точку экрана с координатами (**X : integer; Y : integer**). Чтобы задать аргументы операции необходимо в диалоговом окне **Class Specification for Figure Elemen** щелкнуть правой кнопкой на имени операции и в контекстном меню выбрать пункт **Specification...**

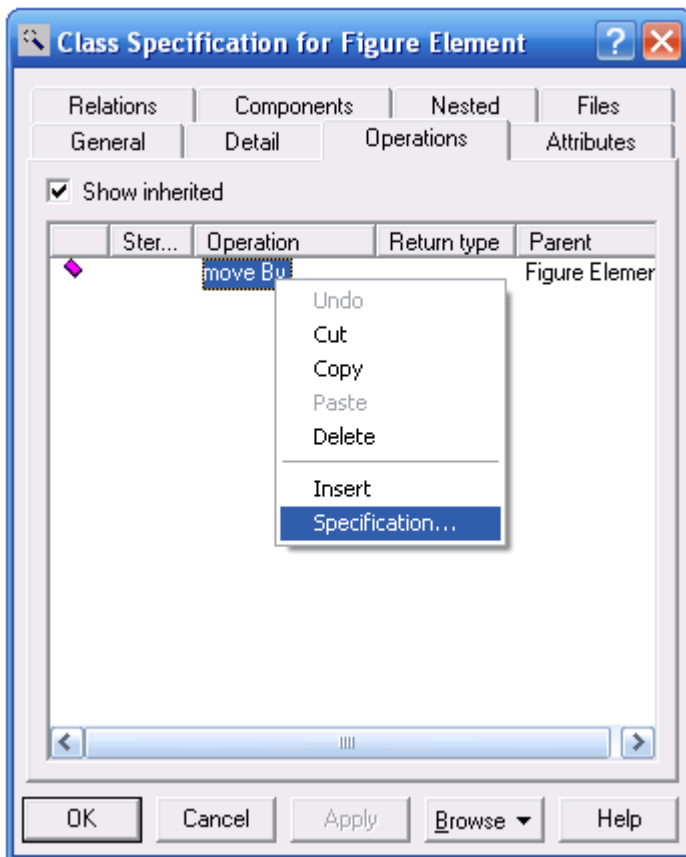


Рисунок 5.14 – Диалоговое окно для задания дополнительных свойств операции **move By**

В этом диалоговом окне на вкладке **Detail** с помощью команды **Insert** контекстного меню задать аргументы **X** типа **Integer** и **Y** типа **Integer**.

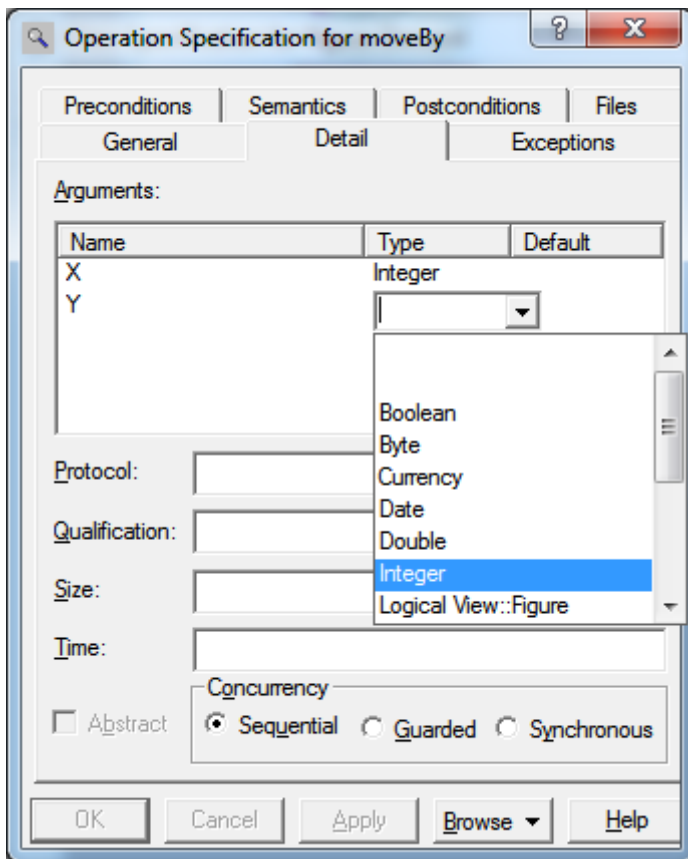


Рисунок 5.15 – Диалоговое окно аргументов операции **move By**

Если все сделано правильно, то после щелчка на операции **move By** в окне диаграммы классов появятся аргументы этой операции как на рис. 5.16.

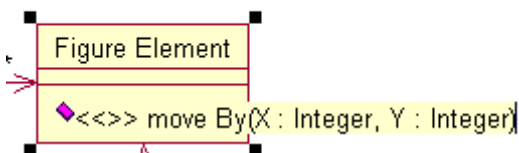


Рисунок 5.16 – Отображение аргументов операции **move By** класса **Figure Element**

3.6. Для класса **Point** задать следующие операции:

- get X();
- get Y();
- set X(int);
- set Y(int);
- move By(int;int).

3.7. Для класса **Line** задать следующие операции:

- get P1();
- get P2();
- set P1(Point);
- set P2(Point);
- move By(int;int).

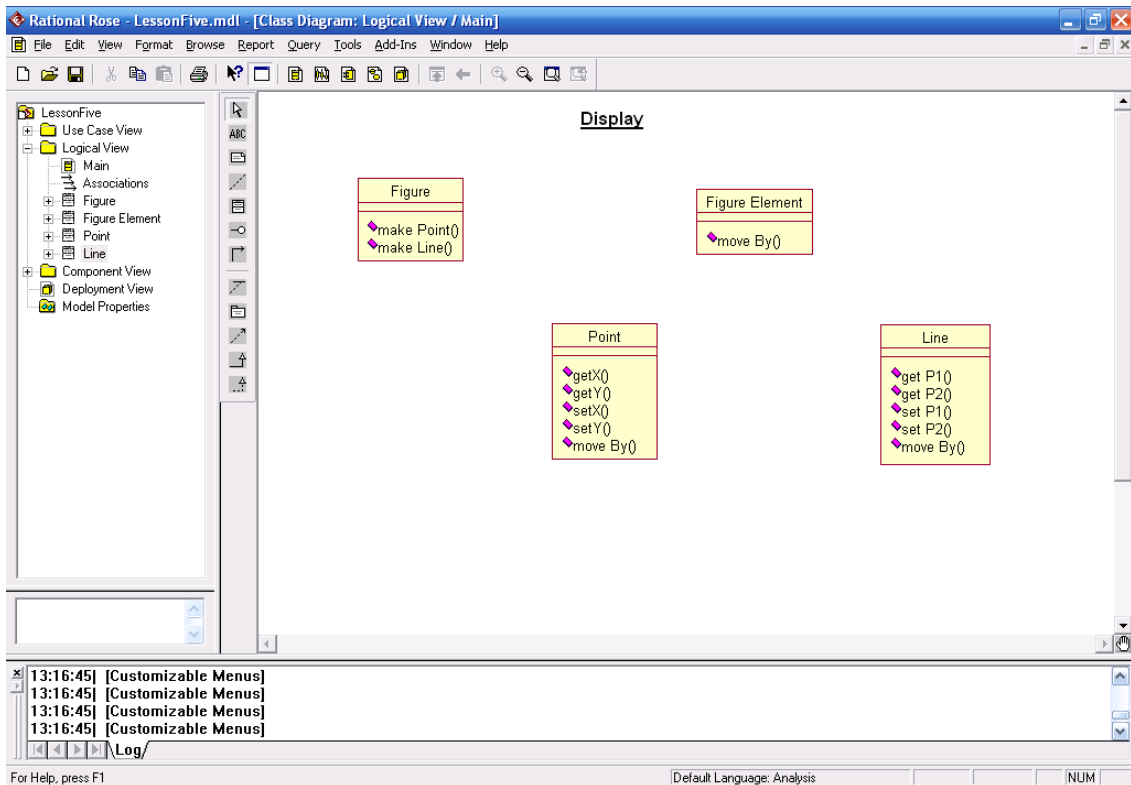


Рисунок 5.17 – Окно диаграммы классов с заданными отношениями

3.8. Установить взаимосвязи между классами как на рис. 5.18.

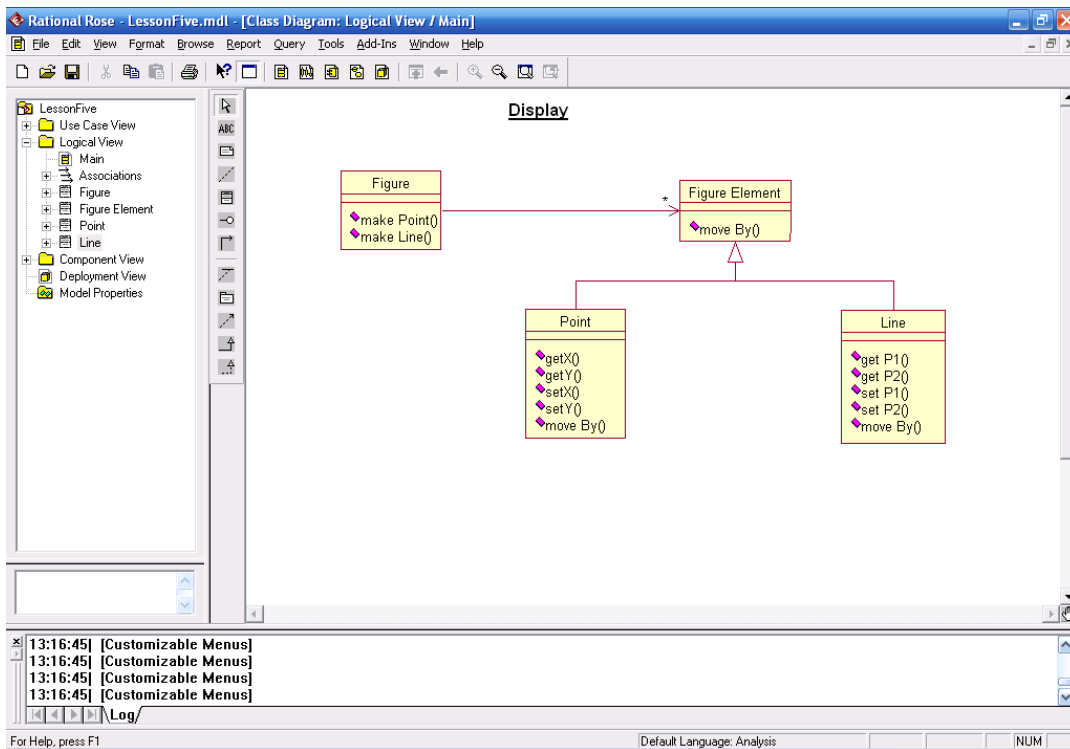


Рисунок 5.18 – Диаграмма классов графического редактора

Задание 4. Сгенерировать программный код графического редактора на языке **ANSI C++**. Порядок выполнения:

4.1. Проверить модель на отсутствие ошибок. Для этого выполнить операцию главного меню: **Tools** → **Check Model** (Инструменты → Проверить модель). Результаты проверки разработанной модели на наличие ошибок отображаются в окне журнала. Прежде чем приступить к генерации текста программного кода разработчику следует добиться устранения всех ошибок и предупреждений, о чем должно свидетельствовать чистое окно журнала (рис. 5.19).

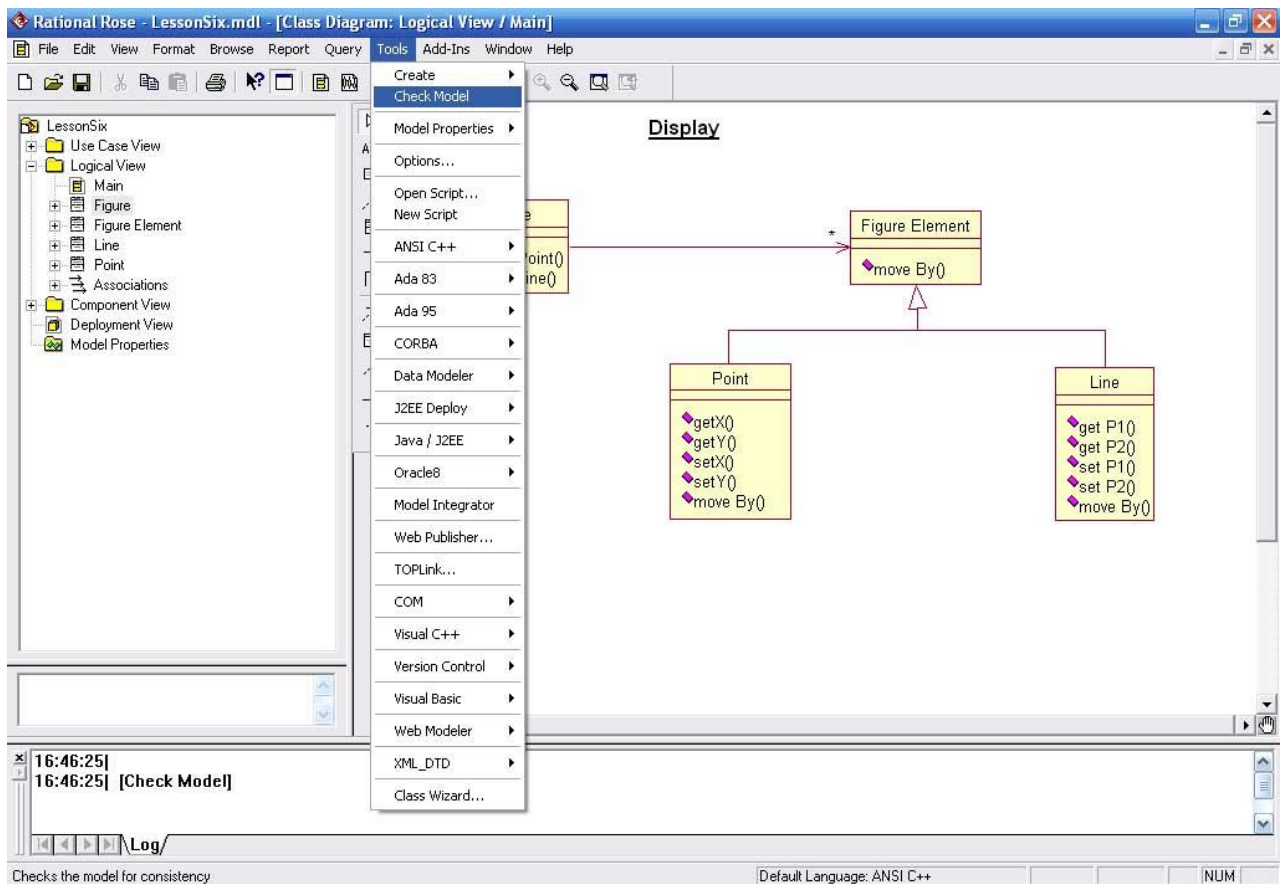



Рисунок 5.19 – Проверка модели на отсутствие ошибок

4.2. Создать компоненты для реализации классов. В браузере проекта предназначено отдельное представление компонентов **Component View**, в котором уже содержится диаграмма компонентов с пустым содержанием и именем по умолчанию **Main** (Главная). Для активизации диаграммы компонентов в браузере проекта раскрыть представление **Component View** и дважды щелкнуть на пиктограмме **Main**. В результате выполнения этих действий появляется новое окно с чистым рабочим листом диаграммы компонентов и специальная панель инструментов, содержащая кнопки для разработки диаграммы компонентов. Далее добавить в область диаграммы компонентов элемент **Component**  и задать ему имя **MainPaint.exe**.

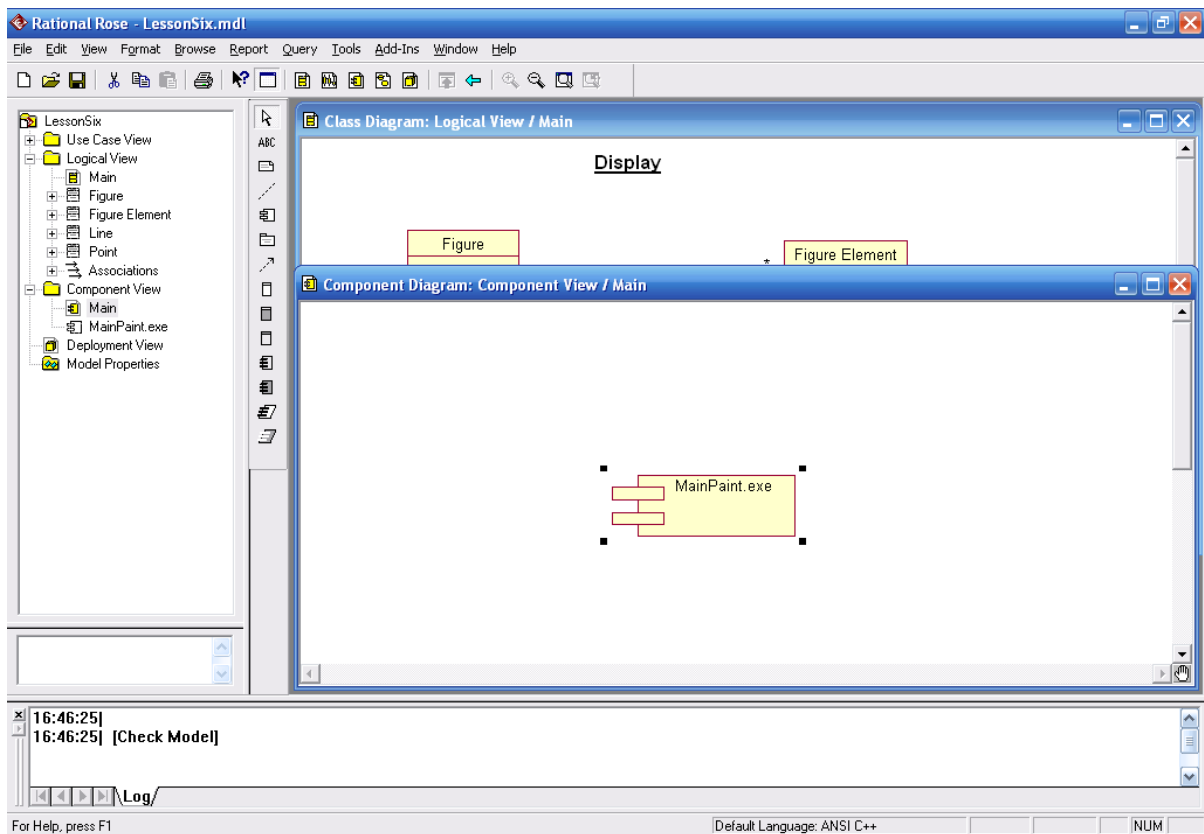


Рисунок 5.20 – Диаграмма компонентов

4.3. Отобразить классы на компоненты. Для этого воспользоваться окном спецификации свойств компонента, открытого на вкладке **Realizes** (Реализует). Для включения реализации класса в данный компонент следует выделить требуемый класс на этой вкладке и выполнить для него операцию контекстного меню **Assign** (Назначить). В результате перед именем класса на этой вкладке появится специальная отметка.

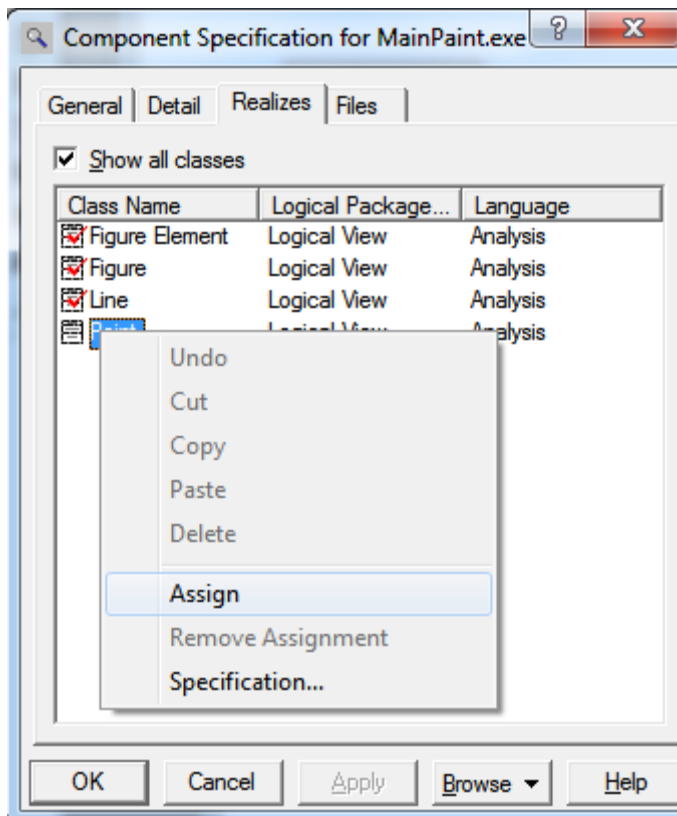


Рисунок 5.21 – Связывание классов с компонентом

4.4 . Выбрать языка программирования для генерации программного кода.

Для выбора языка **ANSI C++** в качестве языка реализации модели следует выполнить операцию главного меню: **Tools** → **Options** (Инструменты → Параметры), в результате чего будет вызвано диалоговое окно настройки параметров модели. Далее на вкладке **Notation** (Нотация) в строке **Default Language** (Язык по умолчанию) из вложенного списка следует выбрать язык **ANSI C++**.

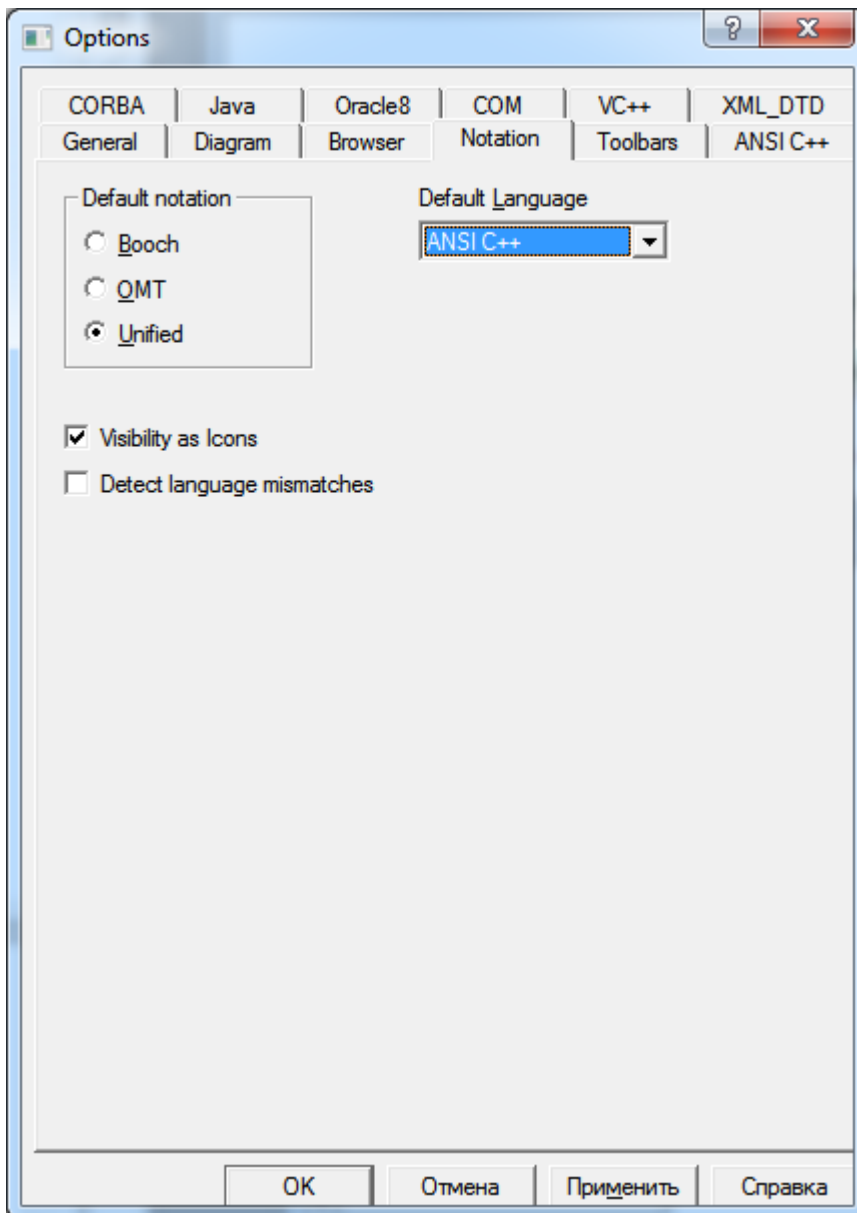


Рисунок 5.22 – Диалоговое окно выбора языка программирования

После выбора языка программирования по умолчанию следует изменить язык реализации каждого из компонентов модели. Для этого в окне диаграммы компонентов с помощью контекстного меню компонента открыть окно спецификации свойств компонента и на вкладке **General** (Общие) в строке **Language** (Язык) из вложенного списка выбрать язык **ANSI C++**.

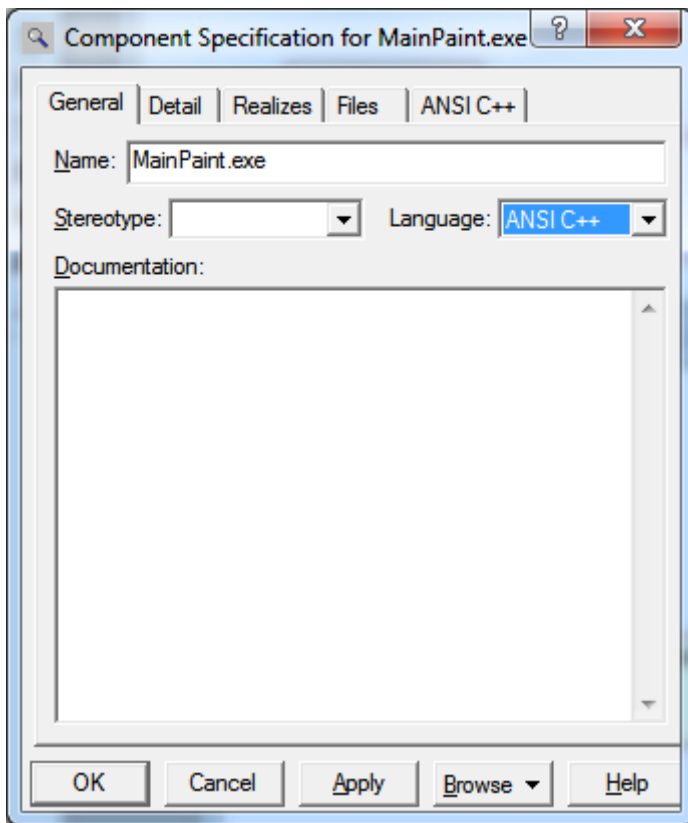


Рисунок 5.23 – Диалоговое окно выбора языка реализации компонентов

4.5. Сгенерировать программный код.

Генерация программного кода в среде IBM Rational Rose возможна для отдельного класса или компонента. Для этого выполнить операцию контекстного меню компонента **ANSI C++** → **Generate Code...** (Язык ANSI C++ → Генерировать код). В результате этого будет открыто диалоговое окно, в котором необходимо указать путь для сохранения сгенерированных файлов, а затем окно с предложением выбора классов для генерации программного кода на выбранном языке программирования.

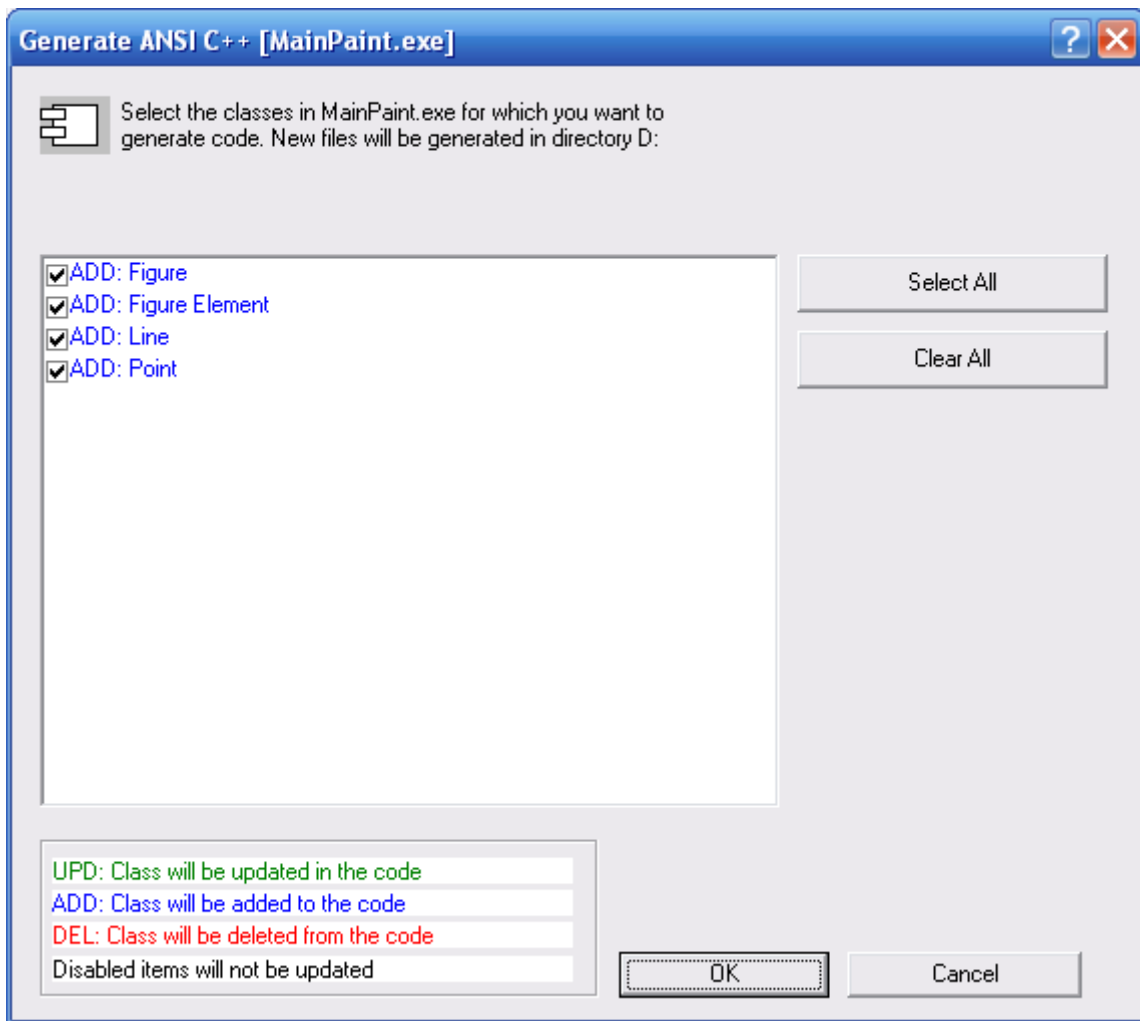


Рисунок 5.24 – Окно выбора классов для генерации программного кода

После генерации программного кода для компонента **MainPaint.exe** каждому классу, реализованному в данном компоненте, будет соответствовать 2 файла с текстом кода на языке **ANSI C++**:

- заголовочный файл с расширением **.h**;
- файл реализации с расширением **.cpp**.

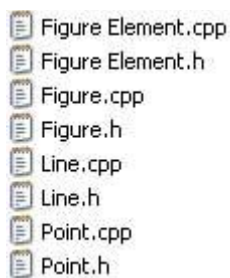


Рисунок 5.25 – Сгенерированные файлы

Контрольные вопросы

1. Какие преимущества предоставляет программа Rational Rose?
2. Как создать диаграмму классов в программе Rational Rose?

3. Какие атрибуты классов Вам известны?
4. Как задать аргументы операции на диаграмме классов?
5. Какие элементы содержит специальная панель инструментов для создания диаграммы классов?
6. Как добавить элемент на специальную панель инструментов?
7. Что называют агрегированием в UML?
8. Что представляет собой диаграмма деятельности?
9. Как создать диаграмму деятельности в программе Rational Rose?
10. Какие элементы содержит специальная панель инструментов для создания диаграммы деятельности?
11. Как задать условие перехода для элемента ветвления?
12. Зачем на диаграмме деятельности используется синхронизационная черта?
13. Какое назначение диаграммы компонентов?
14. Как создать диаграмму компонентов в программе Rational Rose?
15. Опишите последовательность генерации программного кода в Rational Rose.
16. Какие файлы создаются при генерации программного кода на языке ANSI C++?
17. Какие элементы содержит специальная панель инструментов для создания диаграммы компонентов?